CLIMADA documentation

Release 4.1.1

CLIMADA contributors

Feb 21, 2024

CONTENTS

CLIMADA stands for CLIMate ADAptation and is a probabilistic natural catastrophe impact model, that also calculates averted damage (benefit) thanks to adaptation measures of any kind (from grey to green infrastructure, behavioural, etc.).

CLIMADA is primarily developed and maintained by the Weather and Climate Risks Group at ETH Zürich.

If you use CLIMADA for your own scientific work, please reference the appropriate publications according to the *Citation Guide*.

This is the documentation of the CLIMADA core module which contains all functionalities necessary for performing climate risk analysis and appraisal of adaptation options. Modules for generating different types of hazards and other specialized applications can be found in the CLIMADA Petals module.

Jump right in:

- *README*
- *Getting Started*
- *Installation*
- *Overview*
- GitHub Repository
- *Module Reference*

Hint: ReadTheDocs hosts multiple versions of this documentation. Use the drop-down menu on the bottom left to switch versions. `stable` refers to the most recent release, whereas `latest` refers to the latest development version.

CHAPTER

ONE

INTRODUCTION

CLIMADA implements a fully probabilistic risk assessment model. According to the IPCC *[1]*, natural risks emerge through the interplay of climate and weather-related hazards, the exposure of goods or people to this hazard, and the specific vulnerability of exposed people, infrastructure and environment. The unit chosen to measure risk has to be the most relevant one in a specific decision problem, not necessarily monetary units. Wildfire hazard might be measured by burned area, exposure by population or replacement value of homes and hence risk might be expressed as number of affected people in the context of evacuation, or repair cost of buildings in the context of property insurance.

Risk has been defined by the International Organization for Standardization as the "effect of uncertainty on objectives" as the potential for consequences when something of value is at stake and the outcome is uncertain, recognizing the diversity of values. Risk can then be quantified as the combination of the probability of a consequence and its magnitude:

$$\text{risk} = \text{probability} \times \text{severity}$$

In the simplest case, $\times$ stands for a multiplication, but more generally, it represents a convolution of the respective distributions of probability and severity. We approximate the $severity$ as follows:

$$\text{severity} = F(\text{hazard intensity}, \text{exposure}, \text{vulnerability}) = \text{exposure} * f_{\text{imp}}(\text{hazard intensity})$$

where f_{imp} is the impact function which parametrizes to what extent an exposure will be affected by a specific hazard. While 'vulnerability function' is broadly used in the modelers community, we refer to it as 'impact function' to explicitly include the option of opportunities (i.e. negative damages). Using this approach, CLIMADA constitutes a platform to analyse risks of different hazard types in a globally consistent fashion at different resolution levels, at scales from multiple kilometres down to meters, depending on the purpose.

1.1 References

[1] IPCC: Climate Change 2014: Impacts, Adaptation and Vulnerability. Part A: Global and Sectoral Aspects. Contribution of Working Group II to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change, edited by C. B. Field, V. R. Barros, D. J. Dokken, K. J. Mach, M. D. Mastrandrea, T. E. Bilir, M. Chatterjee, K. L. Ebi, Y. O. Estrada, R. C. Genova, B. Girma, E. S. Kissel, A. N. Levy, S. MacCracken, P. R. Mastrandrea, and L. L. White, Cambridge University Press, United Kingdom and New York, NY, USA., 2014.

CHAPTER TWO

GETTING STARTED WITH CLIMADA

This is a short summary of the guides to help you find the information that you need to get started. To learn more about CLIMADA, have a look at the *introduction*. You can also have a look at the paper repository to get an overview of research projects.

2.1 Installation

The first step to getting started is installing CLIMADA. To do so you will need:

1. To get the lastest release from the git repository CLIMADA releases or clone the project with git if you are interested in contributing to the development.
2. To build a conda environment with the dependencies needed by CLIMADA.

For details see the *Installation Instructions*.

If you need to run a model on a computational cluster, have a look at *this guide* to install CLIMADA and run your jobs.

2.2 Programming in Python

It is best to have some basic knowledge of Python programming before starting with CLIMADA. But if you need a quick introduction or reminder, have a look at the short *Python Tutorial*. Also have a look at the python *Python Dos and Don't* guide and at the *Python Performance Guide* for best practice tips.

2.3 Apps for working with CLIMADA

To work with CLIMADA, you will need an application that supports Jupyter Notebooks. There are plugins available for nearly every code editor or IDE, but if you are unsure about which to choose, we recommend JupyterLab, Visual Studio Code or Spyder. It is easy to get confused by all the different softwares and their uses so here is an overview of which tools we use for what:

Use	Tools	Description	Use-ful for
Distribution /manage virtual environment& packages	Recommended:MambaAlt	Install climada, manage & use the climada virtual environment, install packagesAnaconda includes Anaconda navigator, which is a desktop GUI and can be used to launch applications like Jupyter Notebook, Spyder etc.	Climada Users& Developers
IDE(Integrate Development Environment)	Recommended:VSCodeAl many more	Write and run code Useful for Developers:VSCode also has a GUI to commit changes to Git (similar to GitHub Desktop, but in the same place as your code)VSCode test explorer shows results for individual tests & any classes and files containing those tests (folders display a failure or pass icon)	Climada Users& Developers
Git GUI(Graphica User Interface)	GitHub DesktopGitkraken	Provides an interface which keeps track of the branch you're working on, changes you made etc.Allows you to commit changes, push to GitHub etc. without having to use command lineThe code itself is not written using these applications but with your IDE of choice(see above)	Climada Developers
Continuous integration(CI) server	Jenkins	Automatically checks code changes in GitHub repositories, e.g. when you create a pull request for the develop branchPerforms static code analysis using pylintyou don't need to do any installations yourself, this runs automatically when you push new code to GitHubsee *Continuous Integration and GitHub Actions*	Climada Developers

2.4 Tutorials

A good way to start using CLIMADA is to have a look at the Tutorials. The *Main Tutorial* will introduce you the structure of CLIMADA and how to calculate you first impacts, as well as your first appraisal of adaptation options. You can then look at the specific tutorials for each module (for example if you are interested in a specific hazard, like *Tropical Cyclones*, or in learning to *estimate the value of asset exposure*,…).

2.5 Documentation

You can find the documentation of CLIMADA on Read the Docs online. Note that the documentation has several versions: 'latest', 'stable' and explicit version numbers, such as 'v3.1.1', in the url path. 'latest' is created from the 'develop' branch and has the latest changes of the developers, 'stable' from the latest release. For more details about documentation versions, please have a look at here.

2.6 Contributing

If you would like to participate in the development of CLIMADA, carefully read the *Git and Development Guide*. Before making a new feature, discuss with one of the repository admins (Now Chahan, Emmanuel and David). Every new feature or enhancement should be done on a separate branch, which will be merged in the develop branch after being reviewed (see *Checklist*). Finally, the develop branch is merged in the main branch in each CLIMADA release. Each new feature should come with a tutorial and with *Unit and Integration Tests*.

2.7 Other Questions

If you cannot find you answer in the other guides provided here, you can open an issue for somebody to help you.

CHAPTER

THREE

INSTALLATION

The following sections will guide you through the installation of CLIMADA and its dependencies.

> **Attention:** CLIMADA has a complicated set of dependencies that cannot be installed with `pip` alone. Please follow the installation instructions carefully! We recommend to use Conda for creating a suitable software environment to execute CLIMADA.

All following instructions should work on any operating system (OS) that is supported by Conda, including in particular: **Windows**, **macOS**, and **Linux**.

Hint: If you need help with the vocabulary used on this page, refer to the *Glossary*.

3.1 Prerequisites

- Make sure you are using the **latest version** of your OS. Install any outstanding **updates**.
- Free up at least 10 GB of **free storage space** on your machine. Conda and the CLIMADA dependencies will require around 5 GB of free space, and you will need at least that much additional space for storing the input and output data of CLIMADA.
- Ensure a **stable internet connection** for the installation procedure. All dependencies will be downloaded from the internet. Do **not** use a metered, mobile connection!
- Install the Conda environment management system. We highly recommend you use Miniforge, which includes the potent Mamba package manager. Download the installer suitable for your system and follow the respective installation instructions. We do **not** recommend using the `conda` command anymore, rather use `mamba` (see *Conda as Alternative to Mamba*).

Note: When mentioning the terms "terminal" or "command line" in the following, we are referring to the "Terminal" apps on macOS or Linux and the "Miniforge Prompt" on Windows.

3.1.1 Decide on Your Entry Level!

Depening on your level of expertise, we provide two different approaches:

- If you have never worked with a command line, or if you just want to give CLIMADA a try, follow the *simple instructions*.
- If you want to use the very latest development version of CLIMADA or even develop new CLIMADA code, follow the *advanced instructions*.

Both approaches are not mutually exclusive. After successful installation, you may switch your setup at any time.

3.1.2 Notes on the CLIMADA Petals Package

CLIMADA is divided into two packages, CLIMADA Core (climada_python) and CLIMADA Petals (climada_petals). The Core contains all the modules necessary for probabilistic impact, averted damage, uncertainty and forecast calculations. Data for hazard, exposures and impact functions can be obtained from the *CLIMADA Data API*. Hazard and Exposures subclasses are included as demonstrators only.

> **Attention:** CLIMADA Petals is **not** a standalone module and requires CLIMADA Core to be installed!

CLIMADA Petals contains all the modules for generating data (e.g., `TC_Surge`, `WildFire`, `OpenStreeMap`, ...). New modules are developed and tested here. Some data created with modules from Petals is available to download from the *Data API*. This works with just CLIMADA Core installed. CLIMADA Petals can be used to generate additional data of this type, or to have a look at the tutorials for all data types available from the API.

Both *installation approaches* mentioned above support CLIMADA Petals. If you are unsure whether you need Petals, you can install the Core first and later add Petals in both approaches.

3.2 Simple Instructions

These instructions will install the most recent stable version of CLIMADA without cloning its repository.

1. Open the command line. Create a new Conda environment with CLIMADA by executing

   ```
   mamba create -n climada_env -c conda-forge climada
   ```

2. Activate the environment:

   ```
   mamba activate climada_env
   ```

   You should now see `(climada_env)` appear in the beginning of your command prompt. This means the environment is activated.

3. Verify that everything is installed correctly by executing a single test:

   ```
   python -m unittest climada.engine.test.test_impact
   ```

   Executing CLIMADA for the first time will take some time because it will generate a directory tree in your home/user directory. After a while, some text should appear in your terminal. In the end, you should see an "Ok". If so, great! You are good to go.

4. *Optional:* Install CLIMADA Petals into the environment:

```
mamba install -n climada_env -c conda-forge climada-petals
```

3.3 Advanced Instructions

For advanced Python users or developers of CLIMADA, we recommed cloning the CLIMADA repository and installing the package from source.

> **Warning:** If you followed the *Simple Instructions* before, make sure you **either** remove the environment with
>
> ```
> mamba env remove -n climada_env
> ```
>
> before you continue, **or** you use a **different** environment name for the following instructions (e.g. `climada_dev` instead of `climada_env`).

1. If you are using a **Linux** OS, make sure you have `git` installed (Windows and macOS users are good to go once Conda is installed). On Ubuntu and Debian, you may use APT:

   ```
   apt update
   apt install git
   ```

   Both commands will probably require administrator rights, which can be enabled by prepending `sudo`.

2. Create a folder for your code. We will call it **workspace directory**. To make sure that your user can manipulate it without special privileges, use a subdirectory of your user/home directory. Do **not** use a directory that is synchronized by cloud storage systems like OneDrive, iCloud or Polybox!

3. Open the command line and navigate to the workspace directory you created using `cd`. Replace `<path/to/workspace>` with the path of the workspace directory:

   ```
   cd <path/to/workspace>
   ```

4. Clone CLIMADA from its GitHub repository. Enter the directory and check out the branch of your choice. The latest development version will be available under the branch `develop`.

   ```
   git clone https://github.com/CLIMADA-project/climada_python.git
   cd climada_python
   git checkout develop
   ```

5. Create an Conda environment called `climada_env` for installing CLIMADA:

   ```
   mamba create -n climada_env python=3.9
   ```

   Note: CLIMADA can be installed for different Python versions. If you want to use a different version, replace the version specification in the command above with another allowed version.

Supported Version	`3.9`
Allowed Versions	`3.9`, `3.10`, `3.11`

6. Use the default environment specs in `env_climada.yml` to install all dependencies. Then activate the environment:

```
mamba env update -n climada_env -f requirements/env_climada.yml
mamba activate climada_env
```

7. Install the local CLIMADA source files as Python package using `pip`:

```
python -m pip install -e ./
```

Hint: Using a path `./` (referring to the path you are currently located at) will instruct `pip` to install the local files instead of downloading the module from the internet. The `-e` (for "editable") option further instructs `pip` to link to the source files instead of copying them during installation. This means that any changes to the source files will have immediate effects in your environment, and re-installing the module is never required.

8. Verify that everything is installed correctly by executing a single test:

```
python -m unittest climada.engine.test.test_impact
```

Executing CLIMADA for the first time will take some time because it will generate a directory tree in your home/user directory. If this test passes, great! You are good to go.

3.3.1 Install Developer Dependencies (Optional)

Building the documentation and running the entire test suite of CLIMADA requires additional dependencies which are not installed by default. They are also not needed for using CLIMADA. However, if you want to develop CLIMADA, we recommend you install them.

With the `climada_env` activated, enter the workspace directory and then the CLIMADA repository as above. Then, add the `dev` extra specification to the `pip install` command (**mind the quotation marks**, and see also pip install examples):

```
python -m pip install -e "./[dev]"
```

The CLIMADA Python package defines the following extras:

Extra	Includes Dependencies…
`doc`	for building documentation
`test`	for running and evaluating tests
`dev`	combination of `doc` and `test`

For executing the pre-defined test scripts in exactly the same way as they are executed by the automated CI pipeline, you will need `make` to be installed. On macOS and on Linux it is pre-installed. On Windows, it can easily be installed with Conda:

```
mamba install -n climada_env make
```

Instructions for running the test scripts can be found in the *Testing Guide*.

3.3.2 Install CLIMADA Petals (Optional)

If you are unsure whether you need Petals, see the *notes above*.

To install CLIMADA Petals, we assume you have already installed CLIMADA Core with the *advanced instructions* above.

1. Open the command line and navigate to the workspace directory.
2. Clone CLIMADA Petals from its repository. Enter the directory and check out the branch of your choice. The latest development version will be available under the branch `develop`.

```
git clone https://github.com/CLIMADA-project/climada_petals.git
cd climada_petals
git checkout develop
```

3. Update the Conda environment with the specifications from Petals and activate it:

```
mamba env update -n climada_env -f requirements/env_climada.yml
mamba activate climada_env
```

4. Install the CLIMADA Petals package:

```
python -m pip install -e ./
```

3.3.3 JupyterLab

1. Install JupyterLab into the Conda environment:

```
mamba install -n climada_env -c conda-forge jupyterlab
```

2. Make sure that the `climada_env` is activated (see above) and then start JupyterLab:

```
mamba env activate climada_env
jupyter-lab
```

JupyterLab will open in a new window of your default browser.

3.3.4 Visual Studio Code (VSCode)

Basic Setup

1. Download and install VSCode following the instructions on https://code.visualstudio.com/.
2. Install the Python and Jupyter extensions. In the left sidebar, select the “Extensions” symbol, enter “Python” in the search bar and click *Install* next to the “Python” extension. Repeat this process for “Jupyter”.
3. Open a Jupyter Notebook or create a new one. On the top right, click on *Select Kernel*, select *Python Environments…* and then choose the Python interpreter from the `climada_env`.

See the VSCode docs on Python and Jupyter Notebooks for further information.

Hint: Both of the following setup instructions work analogously for Core and Petals. The specific instructions for Petals are shown in square brackets: []

Workspace Setup

Setting up a workspace for the CLIMADA source code is only available for *advanced installations*.

1. Open a new VSCode window. Below *Start*, click *Open…*, select the `climada_python` [`climada_petals`] repository folder in your workspace directory, and click on *Open* on the bottom right.
2. Click *File* > *Save Workspace As…* and store the workspace settings file next to (**not** in!) the `climada_python` [`climada_petals`] folder. This will enable you to load the workspace and all its specific settings in one go.
3. Open the Command Palette by clicking *View* > *Command Palette* or by using the shortcut keys `Ctrl+Shift+P` (Windows, Linux) / `Cmd+Shift+P` (macOS). Start typing "Python: Select Interpreter" and select it from the dropdown menu. If prompted, choose the option to set the interpreter for the workspace, not just the current folder. Then, choose the Python interpreter from the `climada_env`.

For further information, refer to the VSCode docs on Workspaces.

Test Explorer Setup

After you set up a workspace, you might want to configure the test explorer for easily running the CLIMADA test suite within VSCode.

Note: Please install the additional *test dependencies* before proceeding.

1. In the left sidebar, select the "Testing" symbol, and click on *Configure Python Tests*.
2. Select "pytest" as test framework and then select `climada` [`climada_petals`] as the directory containing the test files.
3. Select "Testing" in the Activity Bar on the left or through *View* > *Testing*. The "Test Explorer" in the left sidebar will display the tree structure of modules, files, test classes and individual tests. You can run individual tests or test subtrees by clicking the Play buttons next to them.
4. By default, the test explorer will show test output for failed tests when you click on them. To view the logs for any test, click on *View* > *Output*, and select "Python Test Log" from the dropdown menu in the view that just opened. If there are errors during test discovery, you can see what's wrong in the "Python" output.

For further information, see the VSCode docs on Python Testing.

3.3.5 Spyder

Installing Spyder into the existing Conda environment for CLIMADA might fail depending on the exact versions of dependencies installed. Therefore, we recommend installing Spyder in a *separate* environment, and then connecting it to a kernel in the original `climada_env`.

1. Follow the Spyder installation instructions. You can follow the "Conda" installation instructions. Keep in mind you are using `mamba`, though!
2. Check the version of the Spyder kernel in the new environment:

   ```
   mamba env export -n spyder-env | grep spyder-kernels
   ```

   This will return a line like this:

   ```
   - spyder-kernels=X.Y.Z=<hash>
   ```

Copy the part `spyder-kernels=X.Y.Z` (until the second =) and paste it into the following command to install the same kernel version into the `climada_env`:

```
mamba install -n climada_env spyder-kernels=X.Y.Z
```

3. Obtain the path to the Python interpreter of your `climada_env`. Execute the following commands:

```
mamba activate climada_env
python -c "import sys; print(sys.executable)"
```

Copy the resulting path.

4. Open Spyder through the command line:

```
mamba activate spyder-env
spyder
```

5. Set the Python interpreter used by Spyder to the one of `climada_env`. Select *Preferences* > *Python Interpreter* > *Use the following interpreter* and paste the iterpreter path you copied from the `climada_env`.

3.4 FAQs

Answers to frequently asked questions.

3.4.1 Updating CLIMADA

We recommend keeping CLIMADA up-to-date. To update, follow the instructions based on your *installation type*:

- **Simple Instructions:** Update CLIMADA using `mamba`:

```
mamba update -n climada_env -c conda-forge climada
```

- **Advanced Instructions:** Move into your local CLIMADA repository and pull the latest version of your respective branch:

```
cd <path/to/workspace>/climada_python
git pull
```

Then, update the environment and reinstall the package:

```
mamba env update -n climada_env -f requirements/env_climada.yml
mamba activate climada_env
python -m pip install -e ./
```

The same instructions apply for CLIMADA Petals.

3.4.2 Installing More Packages

You might use CLIMADA in code that requires more packages than the ones readily available in the CLIMADA Conda environment. If so, **prefer installing these packages via Conda**, and only rely on `pip` if that fails. The default channels of Conda sometimes contain outdated versions. Therefore, use the `conda-forge` channel:

```
mamba install -n climada_env -c conda-forge <package>
```

Only if the desired package (version) is not available, go for `pip`:

```
mamba activate climada_env
python -m pip install <package>
```

3.4.3 Verifying Your Installation

If you followed the installation instructions, you already executed a single unit test. This test, however, will not cover all issues that could occur within your installation setup. If you are unsure if everything works as intended, try running all unit tests. This is only available for *advanced setups*! Move into the CLIMADA repository, activate the environment and then execute the tests:

```
cd <path/to/workspace>/climada_python
mamba activate climada_env
python -m unittest discover -s climada -p "test*.py"
```

3.4.4 Error: `ModuleNotFoundError`

Something is wrong with the environment you are using. After **each** of the following steps, check if the problem is solved, and only continue if it is **not**:

1. Make sure you are working in the CLIMADA environment:

   ```
   mamba activate climada_env
   ```

2. *Update the Conda environment and CLIMADA*.
3. Conda will notify you if it is not up-to-date. In this case, follow its instructions to update it. Then, repeat the last step and update the environment and CLIMADA (again).
4. Install the missing package manually. Follow the instructions for *installing more packages*.
5. If you reached this point, something is severely broken. The last course of action is to delete your CLIMADA environment:

   ```
   mamba deactivate
   mamba env remove -n climada_env
   ```

   Now repeat the *installation process*.
6. Still no good? Please raise an issue on GitHub to get help.

3.4.5 Logging Configuration

Climada makes use of the standard logging package. By default, the "climada"-`Logger` is detached from `logging.root`, logging to *stdout* with the level set to `WARNING`.

If you prefer another logging configuration, e.g., for using Climada embedded in another application, you can opt out of the default pre-configuration by setting the config value for `logging.climada_style` to `false` in the *configuration file* `climada.conf`.

Changing the logging level can be done in multiple ways:

- Adjust the *configuration file* `climada.conf` by setting a the value of the `global.log_level` property. This only has an effect if the `logging.climada_style` is set to `true` though.
- Set a global logging level in your Python script:

```
import logging
logging.getLogger('climada').setLevel(logging.ERROR)  # to silence all warnings
```

- Set a local logging level in a context manager:

```
from climada.util import log_level
with log_level(level="INFO"):
    # This also emits all info log messages
    foo()

# Default logging level again
bar()
```

All three approaches can also be combined.

3.4.6 Conda as Alternative to Mamba

We experienced several issues with the default `conda` package manager lately. This is likely due to the large dependency set of CLIMADA, which makes solving the environment a tedious task. We therefore switched to the more performant `mamba` and recommend using it.

Caution: In theory, you could also use an Anaconda or Miniconda distribution and replace every `mamba` command in this guide with `conda`. In practice, however, `conda` is often unable to solve an environment that `mamba` solves without issues in few seconds.

3.4.7 Error: `operation not permitted`

Conda might report a permission error on macOS Mojave. Carefully follow these instructions: https://github.com/conda/conda/issues/8440#issuecomment-481167572

3.4.8 No `impf_TC` Column in `GeoDataFrame`

This may happen when a demo file from CLIMADA was not updated after the change in the impact function naming pattern from `if_` to `impf_` when CLIMADA v2.2.0 was released. Execute

```
mamba activate climada_env
python -c "import climada; climada.setup_climada_data(reload=True)"
```

3.5 The What Now? (Glossary)

You might have become confused about all the names thrown at you. Let's clear that up:

Terminal, Command Line
: A text-only program for interacting with your computer (the old fashioned way). If you are using Miniforge on Windows, the program is called "Miniforge Prompt".

Conda
: A cross-platform package management system. Comes in different varieties (distributions).

Mamba
: The faster reimplementation of the `conda` package manager.

Environment (Programming)
: A setup where only a specific set of modules and programs can interact. This is especially useful if you want to install programs with mutually incompatible requirements.

pip
: The Python package installer.

git
: A popular version control software for programming code (or any text-based set of files).

GitHub
: A website that publicly hosts git repositories.

git Repository
: A collection of files and their entire revision/version history, managed by git.

Cloning
: The process and command (`git clone`) for downloading a git repository.

IDE
: Integrated Development Environment. A fancy source code editor tailored for software development and engineering.

CHAPTER

FOUR

USING CLIMADA ON THE EULER CLUSTER (ETH INTERNAL)

4.1 Access to Euler

See https://scicomp.ethz.ch/wiki/Getting_started_with_clusters for details on how to register at and get started with Euler.

For all steps below, first enter the Cluster via SSH.

4.2 Installation and working directories

Please, get familiar with the various Euler storage options: https://scicomp.ethz.ch/wiki/Storage_systems. As a general rule: use `/cluster/project` for installation and `/cluster/work` for data processing.

For ETH WCR group members, the suggested installation and working directories are `/cluster/project/climate/$USER` and `/cluster/work/climate/$USER` respectively. You may have to create the installation directory:

```
mkdir -p /cluster/project/climate/$USER \
         /cluster/work/climate/$USER
```

4.3 Pre-installed version of Climada

Climada is pre-installed and available in the default pip environment of Euler.

4.3.1 1. Load dependencies

Use the new software stack. Unless you have already done so, run `set_software_stack.sh new`.

```
module load gcc/8.2.0 python/3.10.4 hdf5/1.10.1 gdal/3.4.3 geos/3.9.1 proj/8.2.1␣
↪libspatialindex/1.9.3 netcdf/4.6.3 eccodes/2.31.0 zlib/1.2.9 libtiff/4.2.0 sqlite/3.
↪35.5
```

You need to execute this every time you login to Euler before Climada can be used. To safe yourself from doing it manually, append these lines to the ~/.bashrc script, which is automatically executed upon logging in to Euler.

4.3.2 2. Check installation

```
python -c 'import climada; print(climada.__file__)'
```

should output something like this:

```
/cluster/apps/nss/gcc-8.2.0/python/3.10.4/x86_64/lib64/python3.10/site-packages/
↪climada/__init__.py
```

4.3.3 3. Adjust the Climada configuration

Edit a configuration file according to your needs (see *Guide_Configuration*). Create a climada.conf file e.g., in /cluster/home/$USER/.config with the following content:

```
{
    "local_data": {
        "system": "/cluster/work/climate/USERNAME/climada/data",
        "demo": "/cluster/project/climate/USERNAME/climada/data/demo",
        "save_dir": "/cluster/work/climate/USERNAME/climada/results"
    }
}
```

(Replace USERNAME with your nethz-id.)

4.3.4 4. Run a job

Please see the docs at https://slurm.schedmd.com/ on how to use the `slurm` batch system and the Wiki https://scicomp.ethz.ch/wiki/Transition_from_LSF_to_Slurm for a mapping of `lsf` commands to their `slurm` equivalents.

```
cd /cluster/work/climate/$USER  # change to the working directory
sbatch [slurm options*] --wrap 'python climada_job_script.py'  # submit the job
```

4.4 Working with Git branches

If the Climada version of the default installation is not according to your needs, you can install Climada from a local Git repository.

4.4.1 1. Load dependencies

See *Load dependencies* above.

4.4.2 2. Create installation environment

```
python -m venv --system-site-packages /cluster/project/climate/$USER/climada_venv
```

4.4.3 3. Checkout sources

```
cd /cluster/project/climate/$USER
git clone https://github.com/CLIMADA-project/climada_python.git
cd climada_python
git checkout develop  # i.e., your branch of interest
```

4.4.4 4. Pip install Climada

```
source /cluster/project/climate/$USER/climada_venv/bin/activate
pip install -e /cluster/project/climate/$USER/climada_python
```

4.4.5 5. Check installation

```
cd /cluster/work/climate/$USER
python -c 'import climada; print(climada.__file__)'
```

should output exactly this (with explicit $USER):

```
/cluster/project/climate/$USER/climada_python/climada/__init__.py
```

4.4.6 6. Adjust the Climada configuration

See *Adjust the Climada configuration* above.

4.4.7 7. Run a job

See *Run a job* above.

4.5 Fallback: Conda

If Climada cannot be installed through pip because of changed dependency requirements, there is still the possibility to install Climada through the Conda environment.

> **WARNING:** This approach is highly discouraged, as it imposes a heavy and mostly unnecessary burden on the file system of the cluster.

4.5.1 Installation

1. Conda

Download or update to the latest version of Miniconda. Installation is done by execution of the following steps:

```
cd /cluster/project/climate/USERNAME
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
miniconda3/bin/conda init
rm Miniconda3-latest-Linux-x86_64.sh
```

During the installation process of Miniconda, you are prompted to set the working directory according to your choice. Set it to `/cluster/project/climate/USERNAME/miniconda3`. Once the installation has finished, log out of Euler and in again. The command prompt should be preceded by `(base)`, indicating that the installation was a success and that you login in into conda's base environment by default.

2. Checkout sources

See *Checkout sources* above.

3. Climada environment

Create the conda environment:

```
cd /cluster/project/climate/USERNAME/climada_python
conda env create -f requirements/env_climada.yml --name climada_env
conda env update -n climada_env -f requirements/env_developer.yml

conda activate climada_env
conda install conda-build
conda develop .
```

4. Adjust the Climada configuration

See *Adjust the Climada configuration* above.

5. Climada scripts

Create a bash script for executing python scripts in the climada environment, `climadajob.sh`:

```
#!/bin/bash
PYTHON_SCRIPT=$1
shift
. ~/.bashrc
conda activate climada_env
python $PYTHON_SCRIPT $@
echo $PYTHON_SCRIPT completed
```

Make it executable:

```
chmod +x climadajob.sh
```

Create a python script that executes climada code, e.g., `climada_smoke_test.py`:

```
import sys
from climada import CONFIG, SYSTEM_DIR
from climada.util.test.test_finance import TestNetpresValue
TestNetpresValue().test_net_pres_val_pass()
print(SYSTEM_DIR)
print(CONFIG.local_data.save_dir.str())
print("the script ran with arguments", sys.argv)
```

6. Run a job

With the scripts from above you can submit the python script as a job like this:

```
sbatch [slurm options] --wrap "/path/to/climadajob.sh /path/to/climada_smoke_test.py␣
↪arg1 arg2"
```

After the job has finished the slurm output file should look something like this:

```
/cluster/work/climate/USERNAME/climada/data
/cluster/work/climate/USERNAME/climada/results
the script ran with arguments ['/path/to/climada_smoke_test.py', 'arg1' 'arg2']
python_script.sh completed
```

Please see the docs at https://slurm.schedmd.com/ on how to use the `slurm` batch system and the Wiki https://scicomp.ethz.ch/wiki/Transition_from_LSF_to_Slurm for a mapping of `lsf` commands to their `slurm` equivalents.

4.5.2 Deinstallation

1. Conda

Remove the miniconda3 directory from the installation directory:

```
rm -rf /cluster/project/climate/USERNAME/miniconda3/
```

Delete the conda related parts from `/cluster/home/USERNAME/.bashrc`, i.e., everything between

`# >>> conda initialize >>>`
and
`# <<< conda initialize <<<`

2. Climada

Remove the climada sources and config file:

```
rm -rf /cluster/project/climate/USERNAME/climada_python
rm -f /cluster/home/USERNAME/climada.conf /cluster/home/USERNAME/*/climada.conf
```

4.6 Run Jupyter Notebook on Euler

It is possible to run a Jupyter Notebook on Euler within a JupytherHub instance running as an interactive slurm job. See the documentation https://scicomp.ethz.ch/wiki/JupyterHub.

For using climada inside the jupyter notebook, create a `.jupyterlabrc` file in your Euler home directory with the following content:

```
module purge
module load StdEnv gcc/8.2.0 python_gpu/3.10.4 eth_proxy r/4.2.2 julia/1.8.5 hdf5/1.
↪10.1 gdal/3.4.3 geos/3.9.1 proj/8.2.1 libspatialindex/1.9.3 netcdf/4.6.3 eccodes/2.
↪31.0 zlib/1.2.9 libtiff/4.2.0 sqlite/3.35.5
```

Then login to https://jupyter.euler.hpc.ethz.ch/ and start a JupyterLab server.

4.6.1 Using a virtual environment in a Jupyter notebook

By default the pre-installed climada version is running in your notebooks. If you want to use climada from source you can simply install a python kernel from the `climada_venv` environment, see *Working with Git branches*

Install an IPyhton-kernel:

```
source /cluster/project/climate/$USER/climada_venv/bin/activate
python -m ipykernel install --user --name climada_venv
```

Start a new JupyterLab server, the `climada_venv` kernel should appear in the list of available kernels in JupyterHub.

4.7 Trouble shooting

4.7.1 1. Python Module not found or available

- Make sure your python environment is activated.
- Run `pip install --upgrade MISSING_MODULE`.

4.7.2 2. Upgrading from Python 3.9 to 3.10

Virtual environments created with Python 3.9 are i.g. not working for Python 3.10. In particular Python kernels from 3.9 environments will fail to connect in a Jupyter notebook on https://jupyter.euler.hpc.ethz.ch/.

- It's suggested to create new environments and remove the old kernels from `~/.local/share/jupyter/kernels/`.

CHAPTER

FIVE

SOFTWARE DOCUMENTATION PER PACKAGE

5.1 climada.engine package

5.1.1 climada.engine.unsequa package

climada.engine.unsequa.calc_base module

class climada.engine.unsequa.calc_base.**Calc**

Bases: object

Base class for uncertainty quantification

Contains the generic sampling and sensitivity methods. For computing the uncertainty distribution for specific CLIMADA outputs see the subclass CalcImpact and CalcCostBenefit.

_input_var_names

Names of the required uncertainty variables.

Type

tuple(str)

_metric_names

Names of the output metrics.

Type

tuple(str)

Notes

Parallelization logics: for computation of the uncertainty users may specify a number N of processes on which to perform the computations in parallel. Since the computation for each individual sample of the input parameters is independent of one another, we implemented a simple distribution on the processes.

1. The samples are divided in N equal sub-sample chunks
2. Each chunk of samples is sent as one to a node for processing

Hence, this is equivalent to the user running the computation N times, once for each sub-sample. Note that for each process, all the input variables must be copied once, and hence each parallel process requires roughly the same amount of memory as if a single process would be used.

This approach differs from the usual parallelization strategy (where individual samples are distributed), because each sample requires the entire input data. With this method, copying data between processes is reduced to a minimum.

Parallelization is currently not available for the sensitivity computation, as this requires all samples simoultenaously in the current implementation of the SaLib library.

__init__()

Empty constructor to be overwritten by subclasses

check_distr()

Log warning if input parameters repeated among input variables

Return type
True.

property input_vars

Uncertainty variables

Returns
All uncertainty variables associated with the calculation

Return type
tuple(UncVar)

property distr_dict

Dictionary of the input variable distribution

Probabilitiy density distribution of all the parameters of all the uncertainty variables listed in self.InputVars

Returns
distr_dict – Dictionary of all probability density distributions.

Return type
dict(sp.stats objects)

est_comp_time(*n_samples*, *time_one_run*, *processes=None*)

Estimate the computation time

Parameters

- **n_samples** (*int/float*) – The total number of samples
- **time_one_run** (*int/float*) – Estimated computation time for one parameter set in seconds
- **pool** (*pathos.pool, optional*) – pool that would be used for parallel computation. The default is None.

Return type
Estimated computation time in secs.

make_sample(*N*, *sampling_method='saltelli'*, *sampling_kwargs=None*)

Make samples of the input variables

For all input parameters, sample from their respective distributions using the chosen sampling_method from SALib. https://salib.readthedocs.io/en/latest/api.html

This sets the attributes unc_output.samples_df, unc_output.sampling_method, unc_output.sampling_kwargs.

Parameters

- **N** (*int*) – Number of samples as used in the sampling method from SALib
- **sampling_method** (*str, optional*) – The sampling method as defined in SALib. Possible choices: 'saltelli', 'fast_sampler', 'latin', 'morris', 'dgsm', 'ff' https://salib.readthedocs.io/en/latest/api.html The default is 'saltelli'.

- **sampling_kwargs** (*kwargs, optional*) – Optional keyword arguments passed on to the SALib sampling_method. The default is None.

Returns
unc_output – Uncertainty data object with the samples

Return type
climada.engine.uncertainty.unc_output.UncOutput()

See also:

SALib.sample
sampling methods from SALib SALib.sample

https
//salib.readthedocs.io/en/latest/api.html

sensitivity(*unc_output*, *sensitivity_method='sobol'*, *sensitivity_kwargs=None*)

Compute the sensitivity indices using SALib.

Prior to doing the sensitivity analysis, one must compute the uncertainty (distribution) of the output values (with self.uncertainty()) for all the samples (rows of self.samples_df).

According to Wikipedia, sensitivity analysis is "the study of how the uncertainty in the output of a mathematical model or system (numerical or otherwise) can be apportioned to different sources of uncertainty in its inputs." The sensitivity of each input is often represented by a numeric value, called the sensitivity index. Sensitivity indices come in several forms.

This sets the attributes: sens_output.sensistivity_method sens_output.sensitivity_kwargs sens_output.xxx_sens_df for each metric unc_output.xxx_unc_df

Parameters

- **unc_output** (*climada.engine.unsequa.UncOutput*) – Uncertainty data object in which to store the sensitivity indices
- **sensitivity_method** (*str, optional*) – sensitivity analysis method from SALib.analyse Possible choices:

  'fast', 'rbd_fact', 'morris', 'sobol', 'delta', 'ff'

  The default is 'sobol'. Note that in Salib, sampling methods and sensitivity analysis methods should be used in specific pairs. https://salib.readthedocs.io/en/latest/api.html
- **sensitivity_kwargs** (*dict, optional*) – Keyword arguments of the chosen SALib analyse method. The default is to use SALib's default arguments.

Returns
sens_output – Uncertainty data object with all the sensitivity indices, and all the uncertainty data copied over from unc_output.

Return type
climada.engine.unsequa.UncOutput

climada.engine.unsequa.calc_cost_benefit module

class climada.engine.unsequa.calc_cost_benefit.**CalcCostBenefit**(*haz_input_var: InputVar | Hazard, ent_input_var: InputVar | Entity, haz_fut_input_var: InputVar | Hazard | None = None, ent_fut_input_var: InputVar | Entity | None = None*)

Bases: `Calc`

Cost Benefit uncertainty analysis class

This is the base class to perform uncertainty analysis on the outputs of climada.engine.costbenefit.CostBenefit().

value_unit

Unit of the exposures value

Type

str

haz_input_var

Present Hazard uncertainty variable

Type

InputVar or *Hazard*

ent_input_var

Present Entity uncertainty variable

Type

InputVar or *Entity*

haz_unc_fut_Var

Future Hazard uncertainty variable

Type

InputVar or *Hazard*

ent_fut_input_var

Future Entity uncertainty variable

Type

InputVar or *Entity*

_input_var_names

Names of the required uncertainty variables ('haz_input_var', 'ent_input_var', 'haz_fut_input_var', 'ent_fut_input_var')

Type

tuple(str)

_metric_names

Names of the cost benefit output metrics ('tot_climate_risk', 'benefit', 'cost_ben_ratio',

'imp_meas_present', 'imp_meas_future')

Type
tuple(str)

`__init__`(*haz_input_var: InputVar | Hazard*, *ent_input_var: InputVar | Entity*, *haz_fut_input_var: InputVar | Hazard | None = None*, *ent_fut_input_var: InputVar | Entity | None = None*)

Initialize UncCalcCostBenefit

Sets the uncertainty input variables, the cost benefit metric_names, and the units.

Parameters

- **haz_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.hazard.Hazard Hazard uncertainty variable or Hazard for the present Hazard in climada.engine.CostBenefit.calc
- **ent_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.entity.Entity Entity uncertainty variable or Entity for the present Entity in climada.engine.CostBenefit.calc
- **haz_fut_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.hazard.Hazard, optional Hazard uncertainty variable or Hazard for the future Hazard The Default is None.
- **ent_fut_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.entity.Entity, optional Entity uncertainty variable or Entity for the future Entity in climada.engine.CostBenefit.calc

`uncertainty`(*unc_sample*, *processes=1*, *chunksize=None*, ***cost_benefit_kwargs*)

Computes the cost benefit for each sample in unc_output.sample_df.

By default, imp_meas_present, imp_meas_future, tot_climate_risk, benefit, cost_ben_ratio are computed.

This sets the attributes: unc_output.imp_meas_present_unc_df, unc_output.imp_meas_future_unc_df unc_output.tot_climate_risk_unc_df unc_output.benefit_unc_df unc_output.cost_ben_ratio_unc_df unc_output.unit unc_output.cost_benefit_kwargs

Parameters

- **unc_sample** (*climada.engine.uncertainty.unc_output.UncOutput*) – Uncertainty data object with the input parameters samples
- **processes** (*int, optional*) – Number of CPUs to use for parralel computations. The default is 1 (not parallel)
- **cost_benefit_kwargs** (*keyword arguments*) – Keyword arguments passed on to climada.engine.CostBenefit.calc()
- **chunksize** (*int, optional*) – Size of the sample chunks for parallel processing. Default is equal to the number of samples divided by the number of processes.

Returns
unc_output – Uncertainty data object in with the cost benefit outputs for each sample and all the sample data copied over from unc_sample.

Return type
climada.engine.uncertainty.unc_output.UncCostBenefitOutput

Raises
`ValueError:` – If no sampling parameters defined, the uncertainty distribution cannot be computed.

Notes

Parallelization logic is described in the base class here *Calc*

See also:

climada.engine.cost_benefit
compute risk and adptation option cost benefits.

climada.engine.unsequa.calc_impact module

class climada.engine.unsequa.calc_impact.**CalcImpact**(*exp_input_var:* InputVar | Exposures, *impf_input_var:* InputVar | ImpactFuncSet, *haz_input_var:* InputVar | Hazard)

Bases: *Calc*

Impact uncertainty caclulation class.

This is the class to perform uncertainty analysis on the outputs of a climada.engine.impact.Impact() object.

rp

List of the chosen return periods.

Type
list(int)

calc_eai_exp

Compute eai_exp or not

Type
bool

calc_at_event

Compute eai_exp or not

Type
bool

value_unit

Unit of the exposures value

Type
str

exp_input_var

Exposure uncertainty variable

Type
InputVar or *Exposures*

impf_input_var

Impact function set uncertainty variable

Type
InputVar if ImpactFuncSet

haz_input_var

Hazard uncertainty variable

Type

InputVar or *Hazard*

_input_var_names

Names of the required uncertainty input variables ('exp_input_var', 'impf_input_var', 'haz_input_var')

Type

tuple(str)

_metric_names

Names of the impact output metrics ('aai_agg', 'freq_curve', 'at_event', 'eai_exp')

Type

tuple(str)

__init__(*exp_input_var:* InputVar | Exposures, *impf_input_var:* InputVar | ImpactFuncSet, *haz_input_var:* InputVar | Hazard)

Initialize UncCalcImpact

Sets the uncertainty input variables, the impact metric_names, and the units.

Parameters

- **exp_input_var** (*climada.engine.uncertainty.input_var.InputVar or climada.entity.Exposure*) – Exposure uncertainty variable or Exposure
- **impf_input_var** (*climada.engine.uncertainty.input_var.InputVar or climada.entity.ImpactFuncSet*) – Impact function set uncertainty variable or Impact function set
- **haz_input_var** (*climada.engine.uncertainty.input_var.InputVar or climada.hazard.Hazard*) – Hazard uncertainty variable or Hazard

uncertainty(*unc_sample*, *rp=None*, *calc_eai_exp=False*, *calc_at_event=False*, *processes=1*, *chunksize=None*)

Computes the impact for each sample in unc_data.sample_df.

By default, the aggregated average impact within a period of 1/frequency_unit (impact.aai_agg) and the excees impact at return periods rp (imppact.calc_freq_curve(self.rp).impact) is computed. Optionally, eai_exp and at_event is computed (this may require a larger amount of memory if the number of samples and/or the number of centroids and/or exposures points is large).

This sets the attributes self.rp, self.calc_eai_exp, self.calc_at_event, self.metrics.

This sets the attributes: unc_output.aai_agg_unc_df, unc_output.freq_curve_unc_df unc_output.eai_exp_unc_df unc_output.at_event_unc_df unc_output.unit

Parameters

- **unc_sample** (*climada.engine.uncertainty.unc_output.UncOutput*) – Uncertainty data object with the input parameters samples
- **rp** (*list(int), optional*) – Return periods in years to be computed. The default is [5, 10, 20, 50, 100, 250].
- **calc_eai_exp** (*boolean, optional*) – Toggle computation of the impact at each centroid location. The default is False.
- **calc_at_event** (*boolean, optional*) – Toggle computation of the impact for each event. The default is False.

- **processes** (*int, optional*) – Number of CPUs to use for parralel computations. The default is 1 (not parallel)
- **chunksize** (*int, optional*) – Size of the sample chunks for parallel processing. Default is equal to the number of samples divided by the number of processes.

Returns
unc_output – Uncertainty data object with the impact outputs for each sample and all the sample data copied over from unc_sample.

Return type
climada.engine.uncertainty.unc_output.UncImpactOutput

Raises
`ValueError:` – If no sampling parameters defined, the distribution cannot be computed.

Notes

Parallelization logic is described in the base class here `Calc`

See also:

`climada.engine.impact`
compute impact and risk.

climada.engine.unsequa.input_var module

class `climada.engine.unsequa.input_var.`**`InputVar`**(*func: callable*, *distr_dict: Dict*)

Bases: `object`

Input variable for the uncertainty analysis

An uncertainty input variable requires a single or multi-parameter function. The parameters must follow a given distribution. The uncertainty input variables are the input parameters of the model.

`distr_dict`

Distribution of the uncertainty parameters. Keys are uncertainty parameters names and Values are probability density distribution from the scipy.stats package https://docs.scipy.org/doc/scipy/reference/stats.html

Type
dict

`labels`

Names of the uncertainty parameters (keys of distr_dict)

Type
list

`func`

User defined python fucntion with the uncertainty parameters as keyword arguements and which returns a climada object.

Type
function

Notes

A few default Variables are defined for Hazards, Exposures, Impact Fucntions, Measures and Entities.

Examples

Categorical variable function: LitPop exposures with m,n exponents in [0,5]

```
>>> import scipy as sp
>>> def litpop_cat(m, n):
...     exp = Litpop.from_countries('CHE', exponent=[m, n])
...     return exp
>>> distr_dict = {
...     'm': sp.stats.randint(low=0, high=5),
...     'n': sp.stats.randint(low=0, high=5)
...     }
>>> iv_cat = InputVar(func=litpop_cat, distr_dict=distr_dict)
```

Continuous variable function: Impact function for TC

```
>>> import scipy as sp
>>> def imp_fun_tc(G, v_half, vmin, k, _id=1):
...     intensity = np.linspace(0, 150, num=100)
...     mdd = np.repeat(1, len(intensity))
...     paa = np.array([sigmoid_function(v, G, v_half, vmin, k)
...                     for v in intensity])
...     imp_fun = ImpactFunc(haz_type='TC',
...                          id=_id,
...                          intensity_unit='m/s',
...                          intensity=intensity,
...                          mdd=mdd,
...                          paa=paa)
...     imp_fun.check()
...     impf_set = ImpactFuncSet([imp_fun])
...     return impf_set
>>> distr_dict = {"G": sp.stats.uniform(0.8, 1),
...       "v_half": sp.stats.uniform(50, 100),
...       "vmin": sp.stats.norm(loc=15, scale=30),
...       "k": sp.stats.randint(low=1, high=9)
...       }
>>> iv_cont = InputVar(func=imp_fun_tc, distr_dict=distr_dict)
```

`__init__`(*func: callable*, *distr_dict: Dict*)

Initialize InputVar

Parameters

- **func** (*function*) – Variable defined as a function of the uncertainty parameters
- **distr_dict** (*dict*) – Dictionary of the probability density distributions of the uncertainty parameters, with keys matching the keyword arguments (i.e. uncertainty parameters) of the func function. The distribution must be of type scipy.stats https://docs.scipy.org/doc/scipy/reference/stats.html

`evaluate`(***params*)

Return the value of uncertainty input variable.

By default, the value of the average is returned.

Parameters
****params** (*optional*) – Input parameters will be passed to self.func.

Returns
unc_func(params)** – Output of the uncertainty variable.

Return type
climada object

plot (*figsize=None*)

Plot the distributions of the parameters of the uncertainty variable.

Parameters
figsize (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as:

```
>>> nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3)
>>> figsize = (ncols * FIG_W, nrows * FIG_H)
```

Returns
axes – The figure and axes handle of the plot.

Return type
matplotlib.pyplot.figure, matplotlib.pyplot.axes

static var_to_inputvar (*var*)

Returns an uncertainty variable with no distribution if var is not an InputVar. Else, returns var.

Parameters
var (*climada.uncertainty.InputVar or any other CLIMADA object*)

Returns
var if var is InputVar, else InputVar with var and no distribution.

Return type
InputVar

static haz (*haz_list*, *n_ev=None*, *bounds_int=None*, *bounds_frac=None*, *bounds_freq=None*)

Helper wrapper for basic hazard uncertainty input variable

The following types of uncertainties can be added:

HE: sub-sampling events from the total event set
For each sub-sample, n_ev events are sampled with replacement. HE is the value of the seed for the uniform random number generator.

HI: scale the intensity of all events (homogeneously)
The instensity of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_int

HA: scale the fraction of all events (homogeneously)
The fraction of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_frac

HF: scale the frequency of all events (homogeneously)
The frequency of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_freq

HL: sample uniformly from hazard list
From the provided list of hazard is elements are uniformly sampled. For example, Hazards outputs from dynamical models for different input factors.

If a bounds is None, this parameter is assumed to have no uncertainty.

Parameters

- **haz** (*List of climada.hazard.Hazard*) – The list of base hazard. Can be one or many to uniformly sample from.
- **n_ev** (*int, optional*) – Number of events to be subsampled per sample. Can be equal or larger than haz.size. The default is None.
- **bounds_int** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous intensity scaling. The default is None.
- **bounds_frac** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous fraction scaling. The default is None.
- **bounds_freq** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous frequency scaling. The default is None.

Returns
Uncertainty input variable for a hazard object.

Return type
climada.engine.unsequa.input_var.InputVar

static exp(*exp_list*, *bounds_totval=None*, *bounds_noise=None*)

Helper wrapper for basic exposure uncertainty input variable

The following types of uncertainties can be added:

ET: scale the total value (homogeneously)
The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_totvalue

EN: mutliplicative noise (inhomogeneous)
The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds_noise. EN is the value of the seed for the uniform random number generator.

EL: sample uniformly from exposure list
From the provided list of exposure is elements are uniformly sampled. For example, LitPop instances with different exponents.

If a bounds is None, this parameter is assumed to have no uncertainty.

Parameters

- **exp_list** (*list of climada.entity.exposures.Exposures*) – The list of base exposure. Can be one or many to uniformly sample from.
- **bounds_totval** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous total value scaling.. The default is None.
- **bounds_noise** (*(float, float), optional*) – Bounds of the uniform distribution to scale each exposure point independently. The default is None.

Returns
Uncertainty input variable for an exposure object.

Return type
climada.engine.unsequa.input_var.InputVar

static impfset(*impf_set_list*, *haz_id_dict=None*, *bounds_mdd=None*, *bounds_paa=None*, *bounds_impfi=None*)

Helper wrapper for basic impact function set uncertainty input variable.

One impact function (chosen with haz_type and fun_id) is characterized.

The following types of uncertainties can be added:

MDD: scale the mdd (homogeneously)
The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_mdd

PAA: scale the paa (homogeneously)
The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_paa

IFi: shift the intensity (homogeneously)
The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds_int

IL: sample uniformly from impact function set list
From the provided list of impact function sets elements are uniformly sampled. For example, impact functions obtained from different calibration methods.

If a bounds is None, this parameter is assumed to have no uncertainty.

Parameters

- **impf_set_list** (*list of ImpactFuncSet*) – The list of base impact function set. Can be one or many to uniformly sample from. The impact function ids must identical for all impact function sets.
- **bounds_mdd** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous mdd scaling. The default is None.
- **bounds_paa** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous paa scaling. The default is None.
- **bounds_impfi** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous shift of intensity. The default is None.
- **haz_id_dict** (*dict(), optional*) – Dictionary of the impact functions affected by uncertainty. Keys are hazard types (str), values are a list of impact function id (int). Default is impsf_set.get_ids() i.e. all impact functions in the set

Returns
Uncertainty input variable for an impact function set object.

Return type
climada.engine.unsequa.input_var.InputVar

static ent(*impf_set_list*, *disc_rate*, *exp_list*, *meas_set*, *haz_id_dict*, *bounds_disc=None*, *bounds_cost=None*, *bounds_totval=None*, *bounds_noise=None*, *bounds_mdd=None*, *bounds_paa=None*, *bounds_impfi=None*)

Helper wrapper for basic entity set uncertainty input variable.

Important: only the impact function defined by haz_type and fun_id will be affected by bounds_impfi, bounds_mdd, bounds_paa.

The following types of uncertainties can be added:

DR: value of constant discount rate (homogeneously)
The value of the discounts in each year is sampled uniformly from a distribution with (min, max) = bounds_disc

CO: scale the cost (homogeneously)
The cost of all measures is multiplied by the same number sampled uniformly from a distribution with (min, max) = bounds_cost

ET: scale the total value (homogeneously)
The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_totval

EN: mutliplicative noise (inhomogeneous)
The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds_noise. EN is the value of the seed for the uniform random number generator.

EL: sample uniformly from exposure list
From the provided list of exposure is elements are uniformly sampled. For example, LitPop instances with different exponents.

MDD: scale the mdd (homogeneously)
The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_mdd

PAA: scale the paa (homogeneously)
The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_paa

IFi: shift the intensity (homogeneously)
The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds_int

IL: sample uniformly from impact function set list
From the provided list of impact function sets elements are uniformly sampled. For example, impact functions obtained from different calibration methods.

If a bounds is None, this parameter is assumed to have no uncertainty.

Parameters

- **bounds_disk** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous discount rate scaling. The default is None.
- **bounds_cost** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous cost of all measures scaling. The default is None.
- **bounds_totval** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous total exposure value scaling. The default is None.
- **bounds_noise** (*(float, float), optional*) – Bounds of the uniform distribution to scale each exposure point independently. The default is None.
- **bounds_mdd** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous mdd scaling. The default is None.
- **bounds_paa** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous paa scaling. The default is None.
- **bounds_impfi** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous shift of intensity. The default is None.

- **impf_set_list** (*list of ImpactFuncSet*) – The list of base impact function set. Can be one or many to uniformly sample from. The impact function ids must identical for all impact function sets.
- **disc_rate** (*climada.entity.disc_rates.base.DiscRates*) – The base discount rates.
- **exp_list** (*[climada.entity.exposures.base.Exposure]*) – The list of base exposure. Can be one or many to uniformly sample from.
- **meas_set** (*climada.entity.measures.measure_set.MeasureSet*) – The base measures.
- **haz_id_dict** (*dict*) – Dictionary of the impact functions affected by uncertainty. Keys are hazard types (str), values are a list of impact function id (int).

Returns
Entity uncertainty input variable

Return type
climada.engine.unsequa.input_var.InputVar

static entfut (*impf_set_list*, *exp_list*, *meas_set*, *haz_id_dict*, *bounds_cost=None*, *bounds_eg=None*, *bounds_noise=None*, *bounds_impfi=None*, *bounds_mdd=None*, *bounds_paa=None*)

Helper wrapper for basic future entity set uncertainty input variable.

Important: only the impact function defined by haz_type and fun_id will be affected by bounds_impfi, bounds_mdd, bounds_paa.

The following types of uncertainties can be added:

CO: scale the cost (homogeneously)
The cost of all measures is multiplied by the same number sampled uniformly from a distribution with (min, max) = bounds_cost

EG: scale the exposures growth (homogeneously)
The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_eg

EN: mutliplicative noise (inhomogeneous)
The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds_noise. EN is the value of the seed for the uniform random number generator.

EL: sample uniformly from exposure list
From the provided list of exposure is elements are uniformly sampled. For example, LitPop instances with different exponents.

MDD: scale the mdd (homogeneously)
The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_mdd

PAA: scale the paa (homogeneously)
The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_paa

IFi: shift the impact function intensity (homogeneously)
The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds_impfi

IL: sample uniformly from impact function set list
From the provided list of impact function sets elements are uniformly sampled. For example, impact functions obtained from different calibration methods.

If a bounds is None, this parameter is assumed to have no uncertainty.

Parameters

- **bounds_cost** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous cost of all measures scaling. The default is None.
- **bounds_eg** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous total exposure growth scaling. The default is None.
- **bounds_noise** (*(float, float), optional*) – Bounds of the uniform distribution to scale each exposure point independently. The default is None.
- **bounds_mdd** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous mdd scaling. The default is None.
- **bounds_paa** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous paa scaling. The default is None.
- **bounds_impfi** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous shift of intensity. The default is None.
- **impf_set_list** (*list of ImpactFuncSet*) – The list of base impact function set. Can be one or many to uniformly sample from. The impact function ids must identical for all impact function sets.
- **exp_list** (*[climada.entity.exposures.base.Exposure]*) – The list of base exposure. Can be one or many to uniformly sample from.
- **meas_set** (*climada.entity.measures.measure_set.MeasureSet*) – The base measures.
- **haz_id_dict** (*dict*) – Dictionary of the impact functions affected by uncertainty. Keys are hazard types (str), values are a list of impact function id (int).

Returns

Entity uncertainty input variable

Return type

climada.engine.unsequa.input_var.InputVar

climada.engine.unsequa.unc_output module

class `climada.engine.unsequa.unc_output.`**`UncOutput`**(*samples_df*, *unit=None*)

Bases: `object`

Class to store and plot uncertainty and sensitivity analysis output data

This is the base class to store uncertainty and sensitivity outputs of an analysis done on climada.engine.impact.Impact() or climada.engine.costbenefit.CostBenefit() object.

`samples_df`

Values of the sampled uncertainty parameters. It has n_samples rows and one column per uncertainty parameter.

Type

pandas.DataFrame

`sampling_method`

Name of the sampling method from SAlib. https://salib.readthedocs.io/en/latest/api.html#

Type

str

n_samples

Effective number of samples (number of rows of samples_df)

Type
int

param_labels

Name of all the uncertainty parameters

Type
list

distr_dict

Comon flattened dictionary of all the distr_dict of all input variables. It represents the distribution of all the uncertainty parameters.

Type
dict

problem_sa

The description of the uncertainty variables and their distribution as used in SALib. https://salib.readthedocs.io/en/latest/basics.html.

Type
dict

__init__(*samples_df*, *unit=None*)

Initialize Uncertainty Data object.

Parameters

- **samples_df** (*pandas.DataFrame*) – input parameters samples
- **unit** (*str, optional*) – value unit

order_samples(*by_parameters*)

Function to sort the samples dataframe.

Note: the unc_output.samples_df is ordered inplace.

Parameters
by_parameters (*list[string]*) – List of the uncertainty parameters to sort by (ordering in list is kept)

Return type
None.

get_samples_df()

get_unc_df(*metric_name*)

set_unc_df(*metric_name*, *unc_df*)

get_sens_df(*metric_name*)

set_sens_df(*metric_name*, *sens_df*)

check_salib(*sensitivity_method*)

Checks whether the chosen sensitivity method and the sampling method used to generated self.samples_df respect the pairing recommendation by the SALib package.

https://salib.readthedocs.io/en/latest/api.html

Parameters
sensitivity_method (*str*) – Name of the sensitivity analysis method.

Returns
True if sampling and sensitivity methods respect the recommended pairing.

Return type
bool

property sampling_method

Returns the sampling method used to generate self.samples_df

Returns
Sampling method name

Return type
str

property sampling_kwargs

Returns the kwargs of the sampling method that generate self.samples_df

Returns
Dictionary of arguments for SALib sampling method

Return type
dict

property n_samples

The effective number of samples

Returns
effective number of samples

Return type
int

property param_labels

Labels of all uncertainty input parameters.

Returns
Labels of all uncertainty input parameters.

Return type
list of str

property problem_sa

The description of the uncertainty variables and their distribution as used in SALib. https://salib.readthedocs.io/en/latest/basics.html

Returns
Salib problem dictionary.

Return type
dict

property uncertainty_metrics

Retrieve all uncertainty output metrics names

Returns
unc_metric_list – List of names of attributes containing metrics uncertainty values, without the trailing '_unc_df'

Return type
[str]

property sensitivity_metrics

Retrieve all sensitivity output metrics names

Returns
sens_metric_list – List of names of attributes containing metrics sensitivity values, without the trailing '_sens_df'

Return type
[str]

get_uncertainty(*metric_list=None*)

Returns uncertainty dataframe with values for each sample

Parameters
metric_list (*[str], optional*) – List of uncertainty metrics to consider. The default returns all uncertainty metrics at once.

Returns
Joint dataframe of all uncertainty values for all metrics in the metric_list.

Return type
pandas.DataFrame

See also:

uncertainty_metrics
list of all available uncertainty metrics

get_sensitivity(*salib_si*, *metric_list=None*)

Returns sensitivity index

E.g. For the sensitivity analysis method 'sobol', the choices are ['S1', 'ST'], for 'delta' the choices are ['delta', 'S1'].

For more information see the SAlib documentation: https://salib.readthedocs.io/en/latest/basics.html

Parameters

- **salib_si** (*str*) – Sensitivity index
- **metric_list** (*[str], optional*) – List of sensitivity metrics to consider. The default returns all sensitivity indices at once.

Returns
Joint dataframe of the sensitvity indices for all metrics in the metric_list

Return type
pandas.DataFrame

See also:

sensitvity_metrics
list of all available sensitivity metrics

get_largest_si(*salib_si*, *metric_list=None*, *threshold=0.01*)

Get largest si per metric

Parameters

- **salib_si** (*str*) – The name of the sensitivity index to plot.
- **metric_list** (*list of strings, optional*) – List of metrics to plot the sensitivity. Default is None.
- **threshold** (*float*) – The minimum value a sensitivity index must have to be considered as the largest. The default is 0.01.

Returns
max_si_df – Dataframe with the largest si and its value per metric

Return type
pandas.dataframe

plot_sample(*figsize=None*)

Plot the sample distributions of the uncertainty input parameters

For each uncertainty input variable, the sample distributions is shown in a separate axes.

Parameters
figsize (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as:

```
>>> nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3)
>>> figsize = (ncols * FIG_W, nrows * FIG_H)
```

Raises
ValueError – If no sample was computed the plot cannot be made.

Returns
axes – The axis handle of the plot.

Return type
matplotlib.pyplot.axes

plot_uncertainty(*metric_list=None, orig_list=None, figsize=None, log=False, axes=None, calc_delta=False*)

Plot the uncertainty distribution

For each risk metric, a separate axes is used to plot the uncertainty distribution of the output values obtained over the sampled input parameters.

Parameters

- **metric_list** (*list[str], optional*) – List of metrics to plot the distribution. The default is None.
- **orig_list** (*list[float], optional*) – List of the original (without uncertainty) values for each sub-metric of the mtrics in metric_list. The ordering is identical. The default is None.
- **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as: nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3) figsize = (ncols * FIG_W, nrows * FIG_H)
- **log** (*boolean, optional*) – Use log10 scale for x axis. Default is False.
- **axes** (*matplotlib.pyplot.axes, optional*) – Axes handles to use for the plot. The default is None.
- **calc_delta** (*boolean, optional*) – Adapt x axis label for CalcDeltaImpact unc_output. Default is False.

Raises
ValueError – If no metric distribution was computed the plot cannot be made.

Returns
axes – The axes handle of the plot.

Return type
matplotlib.pyplot.axes

See also:

uncertainty_metrics
list of all available uncertainty metrics

plot_rp_uncertainty(*orig_list=None*, *figsize=(16, 6)*, *axes=None*, *calc_delta=False*)

Plot the distribution of return period uncertainty

Parameters

- **orig_list** (*list[float], optional*) – List of the original (without uncertainty) values for each sub-metric of the metrics in metric_list. The ordering is identical. The default is None.
- **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is (16, 6)
- **axes** (*matplotlib.pyplot.axes, optional*) – Axes handles to use for the plot. The default is None.
- **calc_delta** (*boolean, optional*) – Adapt axis labels for CalcDeltaImpact unc_output. Default is False.

Raises
ValueError – If no metric distribution was computed the plot cannot be made.

Returns
axes – The axis handle of the plot.

Return type
matplotlib.pyplot.axes

plot_sensitivity(*salib_si='S1'*, *salib_si_conf='S1_conf'*, *metric_list=None*, *figsize=None*, *axes=None*, ***kwargs*)

Bar plot of a first order sensitivity index

For each metric, the sensitivity indices are plotted in a separate axes.

This requires that a senstivity analysis was already performed.

E.g. For the sensitivity analysis method 'sobol', the choices are ['S1', 'ST'], for 'delta' the choices are ['delta', 'S1'].

Note that not all sensitivity indices have a confidence interval.

For more information see the SAlib documentation: https://salib.readthedocs.io/en/latest/basics.html

Parameters

- **salib_si** (*string, optional*) – The first order (one value per metric output) sensitivity index to plot. The default is S1.
- **salib_si_conf** (*string, optional*) – The confidence value for the first order sensitivity index to plot. The default is S1_conf.
- **metric_list** (*list of strings, optional*) – List of metrics to plot the sensitivity. If a metric is not found it is ignored.
- **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as:

```
>>> nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3)
>>> figsize = (ncols * FIG_W, nrows * FIG_H)
```

- **axes** (*matplotlib.pyplot.axes, optional*) – Axes handles to use for the plot. The default is None.
- **kwargs** – Keyword arguments passed on to pandas.DataFrame.plot(kind='bar')

Raises

`ValueError` : – If no sensitivity is available the plot cannot be made.

Returns

axes – The axes handle of the plot.

Return type

matplotlib.pyplot.axes

See also:

`sensitvity_metrics`
list of all available sensitivity metrics

`plot_sensitivity_second_order`(*salib_si='S2'*, *salib_si_conf='S2_conf'*, *metric_list=None*, *figsize=None*, *axes=None*, ***kwargs*)

Plot second order sensitivity indices as matrix.

For each metric, the sensitivity indices are plotted in a separate axes.

E.g. For the sensitivity analysis method 'sobol', the choices are ['S2', 'S2_conf'].

Note that not all sensitivity indices have a confidence interval.

For more information see the SAlib documentation: https://salib.readthedocs.io/en/latest/basics.html

Parameters

- **salib_si** (*string, optional*) – The second order sensitivity indexto plot. The default is S2.
- **salib_si_conf** (*string, optional*) – The confidence value for thesensitivity index salib_si to plot. The default is S2_conf.
- **metric_list** (*list of strings, optional*) – List of metrics to plot the sensitivity. If a metric is not found it is ignored. Default is all 1D metrics.
- **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as:

```
>>> nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3)
>>> figsize = (ncols * 5, nrows * 5)
```

- **axes** (*matplotlib.pyplot.axes, optional*) – Axes handles to use for the plot. The default is None.
- **kwargs** – Keyword arguments passed on to matplotlib.pyplot.imshow()

Raises

`ValueError` : – If no sensitivity is available the plot cannot be made.

Returns

axes – The axes handle of the plot.

Return type

matplotlib.pyplot.axes

See also:

sensitvity_metrics
list of all available sensitivity metrics

plot_sensitivity_map(*salib_si='S1'*, ***kwargs*)

Plot a map of the largest sensitivity index in each exposure point

Requires the uncertainty distribution for eai_exp.

Parameters

- **salib_si** (*str, optional*) – The name of the sensitivity index to plot. The default is 'S1'.
- **kwargs** – Keyword arguments passed on to climada.util.plot.geo_scatter_categorical

Raises
`ValueError :` – If no sensitivity data is found, raise error.

Returns
ax – The axis handle of the plot.

Return type
matplotlib.pyplot.axes

See also:

climada.util.plot.geo_scatter_categorical
geographical plot for categorical variable

to_hdf5(*filename=None*)

Save output to .hdf5

Parameters
filename (*str or pathlib.Path, optional*) – The filename with absolute or relative path. The default name is "unc_output + datetime.now() + .hdf5" and the default path is taken from climada.config

Returns
save_path – Path to the saved file

Return type
pathlib.Path

static from_hdf5(*filename*)

Load a uncertainty and uncertainty output data from .hdf5 file

Parameters
filename (*str or pathlib.Path*) – The filename with absolute or relative path.

Returns
unc_output – Uncertainty and sensitivity data loaded from .hdf5 file.

Return type
climada.engine.uncertainty.unc_output.UncOutput

class climada.engine.unsequa.unc_output.**UncCostBenefitOutput**(*samples_df*, *unit*, *imp_meas_present_unc_df*, *imp_meas_future_unc_df*, *tot_climate_risk_unc_df*, *benefit_unc_df*, *cost_ben_ratio_unc_df*, *cost_benefit_kwargs*)

Bases: `UncOutput`

Extension of UncOutput specific for CalcCostBenefit, returned by the uncertainty() method.

`__init__`(*samples_df*, *unit*, *imp_meas_present_unc_df*, *imp_meas_future_unc_df*, *tot_climate_risk_unc_df*, *benefit_unc_df*, *cost_ben_ratio_unc_df*, *cost_benefit_kwargs*)

Constructor

Uncertainty output values from cost_benefit.calc for each sample

Parameters

- **samples_df** (*pandas.DataFrame*) – input parameters samples
- **unit** (*str*) – value unit
- **imp_meas_present_unc_df** (*pandas.DataFrame*) – Each row contains the values of imp_meas_present for one sample (row of samples_df)
- **imp_meas_future_unc_df** (*pandas.DataFrame*) – Each row contains the values of imp_meas_future for one sample (row of samples_df)
- **tot_climate_risk_unc_df** (*pandas.DataFrame*) – Each row contains the values of tot_cliamte_risk for one sample (row of samples_df)
- **benefit_unc_df** (*pandas.DataFrame*) – Each row contains the values of benefit for one sample (row of samples_df)
- **cost_ben_ratio_unc_df** (*pandas.DataFrame*) – Each row contains the values of cost_ben_ratio for one sample (row of samples_df)
- **cost_benefit_kwargs** (*pandas.DataFrame*) – Each row contains the value of cost_benefit for one sample (row of samples_df)

class `climada.engine.unsequa.unc_output.`**`UncImpactOutput`**(*samples_df*, *unit*, *aai_agg_unc_df*, *freq_curve_unc_df*, *eai_exp_unc_df*, *at_event_unc_df*, *coord_df*)

Bases: `UncOutput`

Extension of UncOutput specific for CalcImpact, returned by the uncertainty() method.

`__init__`(*samples_df*, *unit*, *aai_agg_unc_df*, *freq_curve_unc_df*, *eai_exp_unc_df*, *at_event_unc_df*, *coord_df*)

Constructor

Uncertainty output values from impact.calc for each sample

Parameters

- **samples_df** (*pandas.DataFrame*) – input parameters samples
- **unit** (*str*) – value unit
- **aai_agg_unc_df** (*pandas.DataFrame*) – Each row contains the value of aai_aag for one sample (row of samples_df)
- **freq_curve_unc_df** (*pandas.DataFrame*) – Each row contains the values of the impact exceedence frequency curve for one sample (row of samples_df)
- **eai_exp_unc_df** (*pandas.DataFrame*) – Each row contains the values of eai_exp for one sample (row of samples_df)

- **at_event_unc_df** (*pandas.DataFrame*) – Each row contains the values of at_event for one sample (row of samples_df)
- **coord_df** (*pandas.DataFrame*) – Coordinates of the exposure

class climada.engine.unsequa.unc_output.**UncDeltaImpactOutput**(*samples_df*, *unit*, *aai_agg_unc_df*, *freq_curve_unc_df*, *eai_exp_unc_df*, *at_event_initial_unc_df*, *at_event_final_unc_df*, *coord_df*)

Bases: UncOutput

Extension of UncOutput specific for CalcDeltaImpact, returned by the uncertainty() method.

__init__(*samples_df*, *unit*, *aai_agg_unc_df*, *freq_curve_unc_df*, *eai_exp_unc_df*, *at_event_initial_unc_df*, *at_event_final_unc_df*, *coord_df*)

Constructor

Uncertainty output values from impact.calc for each sample

Parameters

- **samples_df** (*pandas.DataFrame*) – input parameters samples
- **unit** (*str*) – value unit
- **aai_agg_unc_df** (*pandas.DataFrame*) – Each row contains the value of aai_aag for one sample (row of samples_df)
- **freq_curve_unc_df** (*pandas.DataFrame*) – Each row contains the values of the impact exceedence frequency curve for one sample (row of samples_df)
- **eai_exp_unc_df** (*pandas.DataFrame*) – Each row contains the values of eai_exp for one sample (row of samples_df)
- **at_event_initial_unc_df** (*pandas.DataFrame*) – Each row contains the values of at_event for one sample (row of samples_df)
- **at_event_final_unc_df** (*pandas.DataFrame*) – Each row contains the values of at_event for one sample (row of samples_df)
- **coord_df** (*pandas.DataFrame*) – Coordinates of the exposure

climada.engine.unsequa.calc_delta_climate module

class climada.engine.unsequa.calc_delta_climate.**CalcDeltaImpact**(*exp_initial_input_var:* InputVar | Exposures, *impf_initial_input_var:* InputVar | ImpactFuncSet, *haz_initial_input_var:* InputVar | Hazard, *exp_final_input_var:* InputVar | Exposures, *impf_final_input_var:* InputVar | ImpactFuncSet, *haz_final_input_var:* InputVar | Hazard)

Bases: *Calc*

Delta Impact uncertainty caclulation class.

This is the class to perform uncertainty analysis on the outputs of a relative change in impact from a (final impact - initial impact) / initial impact. Impact objects are regular climada.engine.impact.Impact() objects. The resulting Delta Impact is a relative change (fraction of the inital impact). The relative change is intuitive to understand in contrast to absolute changes which are hard to understand without knowledge of the absolute initial (baseline) state.

rp

List of the chosen return periods.

Type
list(int)

calc_eai_exp

Compute eai_exp or not

Type
bool

calc_at_event

Compute eai_exp or not

Type
bool

value_unit

Unit of the exposures value

Type
str

exp_input_var

Exposure uncertainty variable

Type
InputVar or *Exposures*

impf_input_var

Impact function set uncertainty variable

Type
InputVar if ImpactFuncSet

haz_input_var

Hazard uncertainty variable

Type

InputVar or *Hazard*

_input_var_names

Names of the required uncertainty input variables ('exp_initial_input_var', 'impf_initial_input_var', 'haz_initial_input_var',

'exp_final_input_var', 'impf_final_input_var', 'haz_final_input_var")

Type

tuple(str)

_metric_names

Names of the impact output metrics ('aai_agg', 'freq_curve', 'at_event', 'eai_exp')

Type

tuple(str)

__init__(*exp_initial_input_var:* InputVar | Exposures, *impf_initial_input_var:* InputVar | ImpactFuncSet, *haz_initial_input_var:* InputVar | Hazard, *exp_final_input_var:* InputVar | Exposures, *impf_final_input_var:* InputVar | ImpactFuncSet, *haz_final_input_var:* InputVar | Hazard)

Initialize UncCalcImpact

Sets the uncertainty input variables, the impact metric_names, and the units.

Parameters

- **exp_initial_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.entity.Exposure Exposure uncertainty variable or Exposure of initial state
- **impf_initital_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.entity.ImpactFuncSet Impact function set uncertainty variable or Impact function set of initial state
- **haz_initial_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.hazard.Hazard Hazard uncertainty variable or Hazard of initial state
- **exp_final_input_var** (*climada.engine.uncertainty.input_var.InputVar or*) – climada.entity.Exposure Exposure uncertainty variable or Exposure of final state
- **impf_final_input_var** (*climada.engine.uncertainty.input_var.InputVar or*) – climada.entity.ImpactFuncSet Impact function set uncertainty variable or Impact function set of final state
- **haz_final_input_var** (*climada.engine.uncertainty.input_var.InputVar or*) – climada.hazard.Hazard Hazard uncertainty variable or Hazard of final state

uncertainty(*unc_sample*, *rp=None*, *calc_eai_exp=False*, *calc_at_event=False*, *processes=1*, *chunksize=None*)

Computes the differential impact between the reference (initial) and future (final) for each sample in unc_data.sample_df.

By default, the aggregated average impact within a period of 1/frequency_unit (impact.aai_agg) and the excess impact at return periods rp (imppact.calc_freq_curve(self.rp).impact) is computed. Optionally, eai_exp and at_event is computed (this may require a larger amount of memory if the number of samples and/or the number of centroids and/or exposures points is large). For all metrics, the impacts are caculated first and the the difference thereof is computed. For example: (impact_final.aai_agg - impact_inital.aai_agg / impact_inital.aai_agg)

This sets the attributes self.rp, self.calc_eai_exp, self.calc_at_event, self.metrics.

This sets the attributes: unc_output.aai_agg_unc_df, unc_output.freq_curve_unc_df unc_output.eai_exp_unc_df unc_output.at_event_unc_df unc_output.unit

Parameters

- **unc_sample** (*climada.engine.uncertainty.unc_output.UncOutput*) – Uncertainty data object with the input parameters samples
- **rp** (*list(int), optional*) – Return periods in years to be computed. The default is [5, 10, 20, 50, 100, 250].
- **calc_eai_exp** (*boolean, optional*) – Toggle computation of the impact at each centroid location. The default is False.
- **calc_at_event** (*boolean, optional*) – Toggle computation of the impact for each event. The default is False.
- **processes** (*int, optional*) – Number of CPUs to use for parralel computations. The default is 1 (not parallel)
- **chunksize** (*int, optional*) – Size of the sample chunks for parallel processing. Default is equal to the number of samples divided by the number of processes.

Returns

unc_output – Uncertainty data object with the delta impact outputs for each sample and all the sample data copied over from unc_sample.

Return type

climada.engine.uncertainty.unc_output.UncImpactOutput

Raises

`ValueError:` – If no sampling parameters defined, the distribution cannot be computed.

Notes

Parallelization logic is described in the base class here `Calc`

See also:

`climada.engine.impact`
compute impact and risk.

5.1.2 climada.engine.calibration_opt module

`climada.engine.calibration_opt.`**`calib_instance`**(*hazard*, *exposure*, *impact_func*, *df_out=Empty DataFrame Columns: [] Index: []*, *yearly_impact=False*, *return_cost='False'*)

calculate one impact instance for the calibration algorithm and write to given DataFrame

Parameters

- **hazard** (*Hazard*)
- **exposure** (*Exposure*)

- **impact_func** (*ImpactFunc*)
- **df_out** (*Dataframe, optional*) – Output DataFrame with headers of columns defined and optionally with first row (index=0) defined with values. If columns "impact", "event_id", or "year" are not included, they are created here. Data like reported impacts or impact function parameters can be given here; values are preserved.
- **yearly_impact** (*boolean, optional*) – if set True, impact is returned per year, not per event
- **return_cost** (*str, optional*) – if not 'False' but any of 'R2', 'logR2', cost is returned instead of df_out

Returns
df_out – DataFrame with modelled impact written to rows for each year or event.

Return type
DataFrame

`climada.engine.calibration_opt.`**`init_impf`**(*impf_name_or_instance, param_dict, df_out=Empty DataFrame Columns: [] Index: [0]*)

create an ImpactFunc based on the parameters in param_dict using the method specified in impf_parameterisation_name and document it in df_out.

Parameters

- **impf_name_or_instance** (*str or ImpactFunc*) – method of impact function parameterisation e.g. 'emanuel' or an instance of ImpactFunc
- **param_dict** (*dict, optional*) – dict of parameter_names and values e.g. {'v_thresh': 25.7, 'v_half': 70, 'scale': 1} or {'mdd_shift': 1.05, 'mdd_scale': 0.8, 'paa_shift': 1, paa_scale': 1}

Returns

- **imp_fun** (*ImpactFunc*) – The Impact function based on the parameterisation
- **df_out** (*DataFrame*) – Output DataFrame with headers of columns defined and with first row (index=0) defined with values. The impact function parameters from param_dict are represented here.

`climada.engine.calibration_opt.`**`change_impf`**(*impf_instance, param_dict*)

apply a shifting or a scaling defined in param_dict to the impact function in impf_istance and return it as a new ImpactFunc object.

Parameters

- **impf_instance** (*ImpactFunc*) – an instance of ImpactFunc
- **param_dict** (*dict*) – dict of parameter_names and values (interpreted as factors, 1 = neutral) e.g. {'mdd_shift': 1.05, 'mdd_scale': 0.8, 'paa_shift': 1, paa_scale': 1}

Returns
ImpactFunc

Return type
The Impact function based on the parameterisation

`climada.engine.calibration_opt.`**`init_impact_data`**(*hazard_type, region_ids, year_range, source_file, reference_year, impact_data_source='emdat', yearly_impact=True*)

creates a dataframe containing the recorded impact data for one hazard type and one area (countries, country or local split)

Parameters

- **hazard_type** (*str*) – default = 'TC', type of hazard 'WS','FL' etc.
- **region_ids** (*str*) – name the region_ids or country names
- **year_range** (*list*) – list containting start and end year. e.g. [1980, 2017]
- **source_file** (*str*)
- **reference_year** (*int*) – impacts will be scaled to this year
- **impact_data_source** (*str, optional*) – default 'emdat', others maybe possible
- **yearly_impact** (*bool, optional*) – if set True, impact is returned per year, not per event

Returns
df_out – Dataframe with recorded impact written to rows for each year or event.

Return type
pd.DataFrame

`climada.engine.calibration_opt.`**`calib_cost_calc`**(*df_out*, *cost_function*)

calculate the cost function of the modelled impact impact_CLIMADA and
the reported impact impact_scaled in df_out

Parameters

- **df_out** (*pd.Dataframe*) – DataFrame as created in calib_instance
- **cost_function** (*str*) – chooses the cost function e.g. 'R2' or 'logR2'

Returns
cost – The results of the cost function when comparing modelled and reported impact

Return type
float

`climada.engine.calibration_opt.`**`calib_all`**(*hazard*, *exposure*, *impf_name_or_instance*, *param_full_dict*, *impact_data_source*, *year_range*, *yearly_impact=True*)

portrait the difference between modelled and reported impacts for all impact functions described in param_full_dict and impf_name_or_instance

Parameters

- **hazard** (*list or Hazard*)
- **exposure** (*list or Exposures*) – list or instance of exposure of full countries
- **impf_name_or_instance** (*string or ImpactFunc*) – the name of a parameterisation or an instance of class ImpactFunc e.g. 'emanuel'
- **param_full_dict** (*dict*) – a dict containing keys used for f_name_or_instance and values which are iterable (lists) e.g. {'v_thresh' : [25.7, 20], 'v_half': [70], 'scale': [1, 0.8]}
- **impact_data_source** (*dict or pd.Dataframe*) – with name of impact data source and file location or dataframe
- **year_range** (*list*)
- **yearly_impact** (*bool, optional*)

Returns
df_result – df with modelled impact written to rows for each year or event.

Return type
pd.DataFrame

climada.engine.calibration_opt.**calib_optimize**(*hazard*, *exposure*, *impf_name_or_instance*, *param_dict*, *impact_data_source*, *year_range*, *yearly_impact=True*, *cost_fucntion='R2'*, *show_details=False*)

portrait the difference between modelled and reported impacts for all impact functions described in param_full_dict and impf_name_or_instance

Parameters

- **hazard** (*list or Hazard*)
- **exposure** (*list or Exposures*) – list or instance of exposure of full countries
- **impf_name_or_instance** (*string or ImpactFunc*) – the name of a parameterisation or an instance of class ImpactFunc e.g. 'emanuel'
- **param_dict** (*dict*) – a dict containing keys used for impf_name_or_instance and one set of values e.g. {'v_thresh': 25.7, 'v_half': 70, 'scale': 1}
- **impact_data_source** (*dict or pd. dataframe*) – with name of impact data source and file location or dataframe
- **year_range** (*list*)
- **yearly_impact** (*bool, optional*)
- **cost_function** (*str, optional*) – the argument for function calib_cost_calc, default 'R2'
- **show_details** (*bool, optional*) – if True, return a tuple with the parameters AND the details of the optimization like success, status, number of iterations etc

Returns
param_dict_result – the parameters with the best calibration results (or a tuple with (1) the parameters and (2) the optimization output)

Return type
dict or tuple

5.1.3 climada.engine.cost_benefit module

class climada.engine.cost_benefit.**CostBenefit**(*present_year: int = 2016*, *future_year: int = 2030*, *tot_climate_risk: float = 0.0*, *unit: str = 'USD'*, *color_rgb: Dict[str, ndarray] | None = None*, *benefit: Dict[str, float] | None = None*, *cost_ben_ratio: Dict[str, float] | None = None*, *imp_meas_present: Dict[str, float | Tuple[float, float] | Impact | ImpactFreqCurve] | None = None*, *imp_meas_future: Dict[str, float | Tuple[float, float] | Impact | ImpactFreqCurve] | None = None*)

Bases: object

Impact definition. Compute from an entity (exposures and impact functions) and hazard.

present_year
present reference year

Type
int

future_year
future year

Type
int

tot_climate_risk
total climate risk without measures

Type
float

unit
unit used for impact

Type
str

color_rgb
color code RGB for each measure.

Type
dict

Key
measure name ('no measure' used for case without measure),

Type
str

Value

Type
np.array

benefit
benefit of each measure. Key: measure name, Value: float benefit

Type
dict

cost_ben_ratio
cost benefit ratio of each measure. Key: measure name, Value: float cost benefit ratio

Type
dict

imp_meas_future
impact of each measure at future or default. Key: measure name ('no measure' used for case without measure), Value: dict with: 'cost' (tuple): (cost measure, cost factor insurance), 'risk' (float): risk measurement, 'risk_transf' (float): annual expected risk transfer, 'efc' (ImpactFreqCurve): impact exceedance freq (optional) 'impact' (Impact): impact instance

Type
dict

imp_meas_present

impact of each measure at present. Key: measure name ('no measure' used for case without measure), Value: dict with: 'cost' (tuple): (cost measure, cost factor insurance), 'risk' (float): risk measurement, 'risk_transf' (float): annual expected risk transfer, 'efc' (ImpactFreqCurve): impact exceedance freq (optional) 'impact' (Impact): impact instance

Type
dict

__init__(*present_year: int = 2016*, *future_year: int = 2030*, *tot_climate_risk: float = 0.0*, *unit: str = 'USD'*, *color_rgb: Dict[str, ndarray] | None = None*, *benefit: Dict[str, float] | None = None*, *cost_ben_ratio: Dict[str, float] | None = None*, *imp_meas_present: Dict[str, float | Tuple[float, float] |* Impact *|* ImpactFreqCurve*] | None = None*, *imp_meas_future: Dict[str, float | Tuple[float, float] |* Impact *|* ImpactFreqCurve*] | None = None*)

Initilization

calc(*hazard*, *entity*, *haz_future=None*, *ent_future=None*, *future_year=None*, *risk_func=<function risk_aai_agg>*, *imp_time_depen=None*, *save_imp=False*, *assign_centroids=True*)

Compute cost-benefit ratio for every measure provided current and, optionally, future conditions. Present and future measures need to have the same name. The measures costs need to be discounted by the user. If future entity provided, only the costs of the measures of the future and the discount rates of the present will be used.

Parameters

- **hazard** (*climada.Hazard*)
- **entity** (*climada.entity*)
- **haz_future** (*climada.Hazard, optional*) – hazard in the future (future year provided at ent_future)
- **ent_future** (*Entity, optional*) – entity in the future. Default is None
- **future_year** (*int, optional*) – future year to consider if no ent_future. Default is None provided. The benefits are added from the entity.exposures.ref_year until ent_future.exposures.ref_year, or until future_year if no ent_future given. Default: entity.exposures.ref_year+1
- **risk_func** (*func optional*) – function describing risk measure to use to compute the annual benefit from the Impact. Default: average annual impact (aggregated).
- **imp_time_depen** (*float, optional*) – parameter which represents time evolution of impact (super- or sublinear). If None: all years count the same when there is no future hazard nor entity and 1 (linear annual change) when there is future hazard or entity. Default is None.
- **save_imp** (*bool, optional*) – Default: False
- **assign_centroids** (*bool, optional*) – indicates whether centroids are assigned to the self.exposures object. Centroids assignment is an expensive operation; set this to `False` to save computation time if the exposures from `ent` and `ent_fut` have already centroids assigned for the respective hazards. Default: True
- **True if Impact of each measure is saved. Default is False.**

combine_measures(*in_meas_names*, *new_name*, *new_color*, *disc_rates*, *imp_time_depen=None*, *risk_func=<function risk_aai_agg>*)

Compute cost-benefit of the combination of measures previously computed by calc with save_imp=True. The benefits of the measures per event are added. To combine with risk transfer options use apply_risk_transfer.

Parameters

- **in_meas_names** (*list(str)*)
- **list with names of measures to combine**
- **new_name** (*str*) – name to give to the new resulting measure new_color (np.array): color code RGB for new measure, e.g. np.array([0.1, 0.1, 0.1])
- **disc_rates** (*DiscRates*) – discount rates instance
- **imp_time_depen** (*float, optional*) – parameter which represents time evolution of impact (super- or sublinear). If None: all years count the same when there is no future hazard nor entity and 1 (linear annual change) when there is future hazard or entity. Default is None.
- **risk_func** (*func, optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).

Return type
climada.CostBenefit

apply_risk_transfer(*meas_name*, *attachment*, *cover*, *disc_rates*, *cost_fix=0*, *cost_factor=1*, *imp_time_depen=None*, *risk_func=<function risk_aai_agg>*)

Applies risk transfer to given measure computed before with saved impact and compares it to when no measure is applied. Appended to dictionaries of measures.

Parameters

- **meas_name** (*str*) – name of measure where to apply risk transfer
- **attachment** (*float*) – risk transfer values attachment (deductible)
- **cover** (*float*) – risk transfer cover
- **cost_fix** (*float*) – fixed cost of implemented innsurance, e.g. transaction costs
- **cost_factor** (*float, optional*) – factor to which to multiply the insurance layer to compute its cost. Default is 1
- **imp_time_depen** (*float, optional*) – parameter which represents time evolution of impact (super- or sublinear). If None: all years count the same when there is no future hazard nor entity and 1 (linear annual change) when there is future hazard or entity. Default is None.
- **risk_func** (*func, optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).

remove_measure(*meas_name*)

Remove computed values of given measure

Parameters
meas_name (*str*) – name of measure to remove

plot_cost_benefit(*cb_list=None*, *axis=None*, ***kwargs*)

Plot cost-benefit graph. Call after calc().

Parameters

- **cb_list** (*list(CostBenefit), optional*) – if other CostBenefit provided, overlay them all. Used for uncertainty visualization.
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for Rectangle matplotlib, e.g. alpha=0.5 (color is set by measures color attribute)

Return type
matplotlib.axes._subplots.AxesSubplot

plot_event_view(*return_per=(10, 25, 100)*, *axis=None*, ***kwargs*)

Plot averted damages for return periods. Call after calc().

Parameters

- **return_per** (*list, optional*) – years to visualize. Default 10, 25, 100
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for bar matplotlib function, e.g. alpha=0.5 (color is set by measures color attribute)

Return type

matplotlib.axes._subplots.AxesSubplot

static plot_waterfall(*hazard*, *entity*, *haz_future*, *ent_future*, *risk_func=<function risk_aai_agg>*, *axis=None*, ***kwargs*)

Plot waterfall graph at future with given risk metric. Can be called before and after calc().

Parameters

- **hazard** (*climada.Hazard*)
- **entity** (*climada.Entity*)
- **haz_future** (*Hazard*) – hazard in the future (future year provided at ent_future). `haz_future` is expected to have the same centroids as `hazard`.
- **ent_future** (*climada.Entity*) – entity in the future
- **risk_func** (*func, optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for bar matplotlib function, e.g. alpha=0.5

Return type

matplotlib.axes._subplots.AxesSubplot

plot_arrow_averted(*axis*, *in_meas_names=None*, *accumulate=False*, *combine=False*, *risk_func=<function risk_aai_agg>*, *disc_rates=None*, *imp_time_depen=1*, ***kwargs*)

Plot waterfall graph with accumulated values from present to future year. Call after calc() with save_imp=True.

Parameters

- **axis** (*matplotlib.axes._subplots.AxesSubplot*) – axis from plot_waterfall or plot_waterfall_accumulated where arrow will be added to last bar
- **in_meas_names** (*list(str), optional*) – list with names of measures to represented total averted damage. Default: all measures
- **accumulate** (*bool, optional)*) – accumulated averted damage (True) or averted damage in future (False). Default: False
- **combine** (*bool, optional*) – use combine_measures to compute total averted damage (True) or just add benefits (False). Default: False
- **risk_func** (*func, optional*) – function describing risk measure given an Impact used in combine_measures. Default: average annual impact (aggregated).
- **disc_rates** (*DiscRates, optional*) – discount rates used in combine_measures

- **imp_time_depen** (*float, optional*) – parameter which represent time evolution of impact used in combine_measures. Default: 1 (linear).
- **kwargs** (*optional*) – arguments for bar matplotlib function, e.g. alpha=0.5

plot_waterfall_accumulated(*hazard*, *entity*, *ent_future*, *risk_func=<function risk_aai_agg>*, *imp_time_depen=1*, *axis=None*, ***kwargs*)

Plot waterfall graph with accumulated values from present to future year. Call after calc() with save_imp=True. Provide same inputs as in calc.

Parameters

- **hazard** (*climada.Hazard*)
- **entity** (*climada.Entity*)
- **ent_future** (*climada.Entity*) – entity in the future
- **risk_func** (*func, optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).
- **imp_time_depen** (*float, optional*) – parameter which represent time evolution of impact used in combine_measures. Default: 1 (linear).
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for bar matplotlib function, e.g. alpha=0.5

Return type
matplotlib.axes._subplots.AxesSubplot

climada.engine.cost_benefit.**risk_aai_agg**(*impact*)

Risk measurement as average annual impact aggregated.

Parameters
impact (*climada.engine.Impact*) – an Impact instance

Return type
float

climada.engine.cost_benefit.**risk_rp_100**(*impact*)

Risk measurement as exceedance impact at 100 years return period.

Parameters
impact (*climada.engine.Impact*) – an Impact instance

Return type
float

climada.engine.cost_benefit.**risk_rp_250**(*impact*)

Risk measurement as exceedance impact at 250 years return period.

Parameters
impact (*climada.engine.Impact*) – an Impact instance

Return type
float

5.1.4 climada.engine.forecast module

class climada.engine.forecast.**Forecast**(*hazard_dict: Dict[str,* Hazard*]*, *exposure:* Exposures, *impact_funcs:* ImpactFuncSet, *haz_model: str = 'NWP'*, *exposure_name: str | None = None*)

Bases: `object`

Forecast definition. Compute an impact forecast with predefined hazard originating from a forecast (like numerical weather prediction models), exposure and impact. Use the calc() method to calculate a forecasted impact. Then use the plotting methods to illustrate the forecasted impacts. By default plots are saved under in a '/forecast/plots' folder in the configurable save_dir in local_data (see climada.util.config) under a name summarizing the Hazard type, haz model name, initialization time of the forecast run, event date, exposure name and the plot title. As the class is relatively new, there might be future changes to the attributes, the methods, and the parameters used to call the methods. It was discovered at some point, that there might be a memory leak in matplotlib even when figures are closed (https://github.com/matplotlib/matplotlib/issues/8519). Due to this reason the plotting functions in this module have the flag close_fig, to close figures within the function scope, which might mitigate that problem if a script runs this plotting functions many times.

run_datetime

initialization time of the forecast model run used to create the Hazard

Type

list of datetime.datetime

event_date

Date on which the Hazard event takes place

Type

datetime.datetime

hazard

List of the hazard forecast with different lead times.

Type

list of CLIMADA Hazard

haz_model

Short string specifying the model used to create the hazard, if possible three big letters.

Type

str

exposure

an CLIMADA Exposures containg values at risk

Type

Exposure

exposure_name

string specifying the exposure (e.g. 'EU'), which is used to name output files.

Type

str

vulnerability

Set of impact functions used in the impact calculation.

Type

ImpactFuncSet

`__init__`(*hazard_dict: Dict[str,* Hazard*]*, *exposure:* Exposures, *impact_funcs:* ImpactFuncSet, *haz_model: str = 'NWP'*, *exposure_name: str | None = None*)

Initialization with hazard, exposure and vulnerability.

Parameters

- **hazard_dict** (*dict*) – Dictionary of the format {run_datetime: Hazard} with run_datetime being the initialization time of a weather forecast run and Hazard being a CLIMADA Hazard derived from that forecast for one event. A probabilistic representation of that one event is possible, as long as the attribute Hazard.date is the same for all events. Several run_datetime:Hazard combinations for the same event can be provided.
- **exposure** (*Exposures*)
- **impact_funcs** (*ImpactFuncSet*)
- **haz_model** (*str, optional*) – Short string specifying the model used to create the hazard, if possible three big letters. Default is 'NWP' for numerical weather prediction.
- **exposure_name** (*str, optional*) – string specifying the exposure (e.g. 'EU'), which is used to name output files. If `None`, the name will be inferred from the Exposures GeoDataframe `region_id` column, using the corresponding name of the region with the lowest ISO 3166-1 numeric code. If that fails, it defaults to `"custom"`.

`ei_exp`(*run_datetime=None*)

Expected impact per exposure

Parameters

run_datetime (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.

Return type

float

`ai_agg`(*run_datetime=None*)

average impact aggregated over all exposures

Parameters

run_datetime (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.

Return type

float

`haz_summary_str`(*run_datetime=None*)

provide a summary string for the hazard part of the forecast

Parameters

run_datetime (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.

Returns

summarizing the most important information about the hazard

Return type

str

`summary_str`(*run_datetime=None*)

provide a summary string for the impact forecast

Parameters
run_datetime (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.

Returns
summarizing the most important information about the impact forecast

Return type
str

lead_time (*run_datetime=None*)
provide the lead time for the impact forecast

Parameters
run_datetime (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.

Returns
the difference between the initialization time of the forecast model run and the date of the event, commenly named lead time

Return type
datetime.timedelta

calc (*force_reassign=False*)
calculate the impacts for all lead times using exposure, all hazards of all run_datetime, and ImpactFunctionSet.

Parameters
force_reassign (*bool, optional*) – Reassign hazard centroids to the exposure for all hazards, default is false.

plot_imp_map (*run_datetime=None, explain_str=None, save_fig=True, close_fig=False, polygon_file=None, polygon_file_crs='epsg:4326', proj=<Derived Projected CRS: +proj=eqc +ellps=WGS84 +a=6378137.0 +lon_0=0.0 +to ...> Name: unknown Axis Info [cartesian]: - E[east]: Easting (unknown) - N[north]: Northing (unknown) - h[up]: Ellipsoidal height (metre) Area of Use: - undefined Coordinate Operation: - name: unknown - method: Equidistant Cylindrical Datum: unknown - Ellipsoid: WGS 84 - Prime Meridian: Greenwich, figsize=(9, 13), adapt_fontsize=True*)
plot a map of the impacts

Parameters

- **run_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.
- **explain_str** (*str, optional*) – Short str which explains type of impact, explain_str is included in the title of the figure. default is 'mean building damage caused by wind'
- **save_fig** (*bool, optional*) – Figure is saved if True, folder is within your configurable save_dir and filename is derived from the method summary_str() (for more details see class docstring). Default is True.
- **close_fig** (*bool, optional*) – Figure not drawn if True. Default is False.
- **polygon_file** (*str, optional*) – Points to a .shp-file with polygons do be drawn as outlines on the plot, default is None to not draw the lines. please also specify the crs in the parameter polygon_file_crs.
- **polygon_file_crs** (*str, optional*) – String of pattern <provider>:<code> specifying the crs. has to be readable by pyproj.Proj. Default is 'epsg:4326'.

- **proj** (*ccrs*) – coordinate reference system used in coordinates The default is ccrs.PlateCarree()
- **figsize** (*tuple*) – figure size for plt.subplots, width, height in inches The default is (9, 13)
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.

Returns
axes

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_hist(*run_datetime=None*, *explain_str=None*, *save_fig=True*, *close_fig=False*, *figsize=(9, 8)*)

plot histogram of the forecasted impacts all ensemble members

Parameters

- **run_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.
- **explain_str** (*str, optional*) – Short str which explains type of impact, explain_str is included in the title of the figure. default is 'total building damage'
- **save_fig** (*bool, optional*) – Figure is saved if True, folder is within your configurable save_dir and filename is derived from the method summary_str() (for more details see class docstring). Default is True.
- **close_fig** (*bool, optional*) – Figure is not drawn if True. Default is False.
- **figsize** (*tuple*) – figure size for plt.subplots, width, height in inches The default is (9, 8)

Returns
axes

Return type
matplotlib.axes.Axes

plot_exceedence_prob(*threshold*, *explain_str=None*, *run_datetime=None*, *save_fig=True*, *close_fig=False*, *polygon_file=None*, *polygon_file_crs='epsg:4326'*, *proj=<Derived Projected CRS: +proj=eqc +ellps=WGS84 +a=6378137.0 +lon_0=0.0 +to ...> Name: unknown Axis Info [cartesian]: - E[east]: Easting (unknown) - N[north]: Northing (unknown) - h[up]: Ellipsoidal height (metre) Area of Use: - undefined Coordinate Operation: - name: unknown - method: Equidistant Cylindrical Datum: unknown - Ellipsoid: WGS 84 - Prime Meridian: Greenwich*, *figsize=(9, 13)*, *adapt_fontsize=True*)

plot exceedence map

Parameters

- **threshold** (*float*) – Threshold of impact unit for which exceedence probability should be plotted.
- **explain_str** (*str, optional*) – Short str which explains threshold, explain_str is included in the title of the figure.
- **run_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.
- **save_fig** (*bool, optional*) – Figure is saved if True, folder is within your configurable save_dir and filename is derived from the method summary_str() (for more details see class docstring). Default is True.

- **close_fig** (*bool, optional*) – Figure not drawn if True. Default is False.
- **polygon_file** (*str, optional*) – Points to a .shp-file with polygons do be drawn as outlines on the plot, default is None to not draw the lines. please also specify the crs in the parameter polygon_file_crs.
- **polygon_file_crs** (*str, optional*) – String of pattern <provider>:<code> specifying the crs. has to be readable by pyproj.Proj. Default is 'epsg:4326'.
- **proj** (*ccrs*) – coordinate reference system used in coordinates The default is ccrs.PlateCarree()
- **figsize** (*tuple*) – figure size for plt.subplots, width, height in inches The default is (9, 13)
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise, the default matplotlib font size is used. Default is True.

Returns
axes

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_warn_map(*polygon_file=None*, *polygon_file_crs='epsg:4326'*, *thresholds='default'*, *decision_level='exposure_point'*, *probability_aggregation=0.5*, *area_aggregation=0.5*, *title='WARNINGS'*, *explain_text='warn level based on thresholds'*, *run_datetime=None*, *proj=<Derived Projected CRS: +proj=eqc +ellps=WGS84 +a=6378137.0 +lon_0=0.0 +to ...> Name: unknown Axis Info [cartesian]: - E[east]: Easting (unknown) - N[north]: Northing (unknown) - h[up]: Ellipsoidal height (metre) Area of Use: - undefined Coordinate Operation: - name: unknown - method: Equidistant Cylindrical Datum: unknown - Ellipsoid: WGS 84 - Prime Meridian: Greenwich*, *figsize=(9, 13)*, *save_fig=True*, *close_fig=False*, *adapt_fontsize=True*)

plot map colored with 5 warning colors for all regions in provided shape file.

Parameters

- **polygon_file** (*str, optional*) – path to shp-file containing warning region polygons
- **polygon_file_crs** (*str, optional*) – String of pattern <provider>:<code> specifying the crs. has to be readable by pyproj.Proj. Default is 'epsg:4326'.
- **thresholds** (*list of 4 floats, optional*) – Thresholds for coloring region in second, third, forth and fifth warning color.
- **decision_level** (*str, optional*) – Either 'exposure_point' or 'polygon'. Default value is 'exposure_point'.
- **probability_aggregation** (*float or str, optional*) – Either a float between [0..1] spezifying a quantile or 'mean' or 'sum'. Default value is 0.5.
- **area_aggregation** (*float or str.*) – Either a float between [0..1] specifying a quantile or 'mean' or 'sum'. Default value is 0.5.
- **run_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.
- **title** (*str, optional*) – Default is 'WARNINGS'.
- **explain_text** (*str, optional*) – Defaut is 'warn level based on thresholds'.
- **proj** (*ccrs*) – coordinate reference system used in coordinates
- **figsize** (*tuple*) – figure size for plt.subplots, width, height in inches The default is (9, 13)

- **save_fig** (*bool, optional*) – Figure is saved if True, folder is within your configurable save_dir and filename is derived from the method summary_str() (for more details see class docstring). Default is True.
- **close_fig** (*bool, optional*) – Figure is not drawn if True. The default is False.
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise, the default matplotlib font size is used. Default is True.

Returns
axes

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_hexbin_ei_exposure(*run_datetime=None*, *figsize=(9, 13)*)

plot the expected impact

Parameters

- **run_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.
- **figsize** (*tuple*) – figure size for plt.subplots, width, height in inches The default is (9, 13)

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

5.1.5 climada.engine.impact module

class climada.engine.impact.**ImpactFreqCurve**(*return_per: ~numpy.ndarray = <factory>*, *impact: ~numpy.ndarray = <factory>*, *unit: str = ''*, *frequency_unit: str = '1/year'*, *label: str = ''*)

Bases: `object`

Impact exceedence frequency curve.

return_per: ndarray

return period

impact: ndarray

impact exceeding frequency

__init__(*return_per: ~numpy.ndarray = <factory>*, *impact: ~numpy.ndarray = <factory>*, *unit: str = ''*, *frequency_unit: str = '1/year'*, *label: str = ''*) → None

unit: str = ''

value unit used (given by exposures unit)

frequency_unit: str = '1/year'

value unit used (given by exposures unit)

label: str = ''

string describing source data

plot(*axis=None*, *log_frequency=False*, ***kwargs*)

Plot impact frequency curve.

Parameters

- **axis** (*matplotlib.axes.Axes, optional*) – axis to use
- **log_frequency** (*boolean, optional*) – plot logarithmioc exceedance frequency on x-axis
- **kwargs** (*dict, optional*) – arguments for plot matplotlib function, e.g. color='b'

Return type
matplotlib.axes.Axes

class climada.engine.impact.**Impact**(*event_id=None*, *event_name=None*, *date=None*, *frequency=None*, *frequency_unit='1/year'*, *coord_exp=None*, *crs='EPSG:4326'*, *eai_exp=None*, *at_event=None*, *tot_value=0*, *aai_agg=0*, *unit=''*, *imp_mat=None*, *haz_type=''*)

Bases: `object`

Impact definition. Compute from an entity (exposures and impact functions) and hazard.

event_id
id (>0) of each hazard event

Type
np.array

event_name
list name of each hazard event

Type
list

date
date if events as integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library)

Type
np.array

coord_exp
exposures coordinates [lat, lon] (in degrees)

Type
np.array

crs
WKT string of the impact's crs

Type
str

eai_exp
expected impact for each exposure within a period of 1/frequency_unit

Type
np.array

at_event
impact for each hazard event

Type
np.array

frequency

frequency of event

Type
np.array

frequency_unit

frequency unit used (given by hazard), default is '1/year'

Type
str

aai_agg

average impact within a period of 1/frequency_unit (aggregated)

Type
float

unit

value unit used (given by exposures unit)

Type
str

imp_mat

matrix num_events x num_exp with impacts. only filled if save_mat is True in calc()

Type
sparse.csr_matrix

haz_type

the hazard type of the hazard

Type
str

__init__(*event_id=None*, *event_name=None*, *date=None*, *frequency=None*, *frequency_unit='1/year'*, *coord_exp=None*, *crs='EPSG:4326'*, *eai_exp=None*, *at_event=None*, *tot_value=0*, *aai_agg=0*, *unit=''*, *imp_mat=None*, *haz_type=''*)

Init Impact object

Parameters

- **event_id** (*np.array, optional*) – id (>0) of each hazard event
- **event_name** (*list, optional*) – list name of each hazard event
- **date** (*np.array, optional*) – date if events as integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library)
- **frequency** (*np.array, optional*) – frequency of event impact for each hazard event
- **frequency_unit** (*np.array, optional*) – frequency unit, default: '1/year'
- **coord_exp** (*np.array, optional*) – exposures coordinates [lat, lon] (in degrees)
- **crs** (*Any, optional*) – Coordinate reference system. CRS instances from `pyproj` and `rasterio` will be transformed into WKT. Other types are not handled explicitly.
- **eai_exp** (*np.array, optional*) – expected impact for each exposure within a period of 1/frequency_unit
- **at_event** (*np.array, optional*) – impact for each hazard event

- **tot_value** (*float, optional*) – total exposure value affected
- **aai_agg** (*float, optional*) – average impact within a period of 1/frequency_unit (aggregated)
- **unit** (*str, optional*) – value unit used (given by exposures unit)
- **imp_mat** (*sparse.csr_matrix, optional*) – matrix num_events x num_exp with impacts.
- **haz_type** (*str, optional*) – the hazard type

`calc`(*exposures*, *impact_funcs*, *hazard*, *save_mat=False*, *assign_centroids=True*)

This function is deprecated, use `ImpactCalc.impact` instead.

`classmethod from_eih`(*exposures*, *hazard*, *at_event*, *eai_exp*, *aai_agg*, *imp_mat=None*)

Set Impact attributes from precalculated impact metrics.

Changed in version 3.3: The `impfset` argument was removed.

Parameters

- **exposures** (*climada.entity.Exposures*) – exposure used to compute imp_mat
- **impfset** (*climada.entity.ImpactFuncSet*) – impact functions set used to compute imp_mat
- **hazard** (*climada.Hazard*) – hazard used to compute imp_mat
- **at_event** (*np.array*) – impact for each hazard event
- **eai_exp** (*np.array*) – expected impact for each exposure within a period of 1/frequency_unit
- **aai_agg** (*float*) – average impact within a period of 1/frequency_unit (aggregated)
- **imp_mat** (*sparse.csr_matrix, optional*) – matrix num_events x num_exp with impacts. Default is None (empty sparse csr matrix).

Returns

impact with all risk metrics set based on the given impact matrix

Return type

climada.engine.impact.Impact

`property tot_value`

Return the total exposure value close to a hazard

Deprecated since version 3.3: Use `climada.entity.exposures.base.Exposures.affected_total_value()` instead.

`transfer_risk`(*attachment*, *cover*)

Compute the risk transfer for the full portfolio. This is the risk of the full portfolio summed over all events. For each event, the transfered risk amounts to the impact minus the attachment (but maximally equal to the cover) multiplied with the probability of the event.

Parameters

- **attachment** (*float*) – attachment per event for entire portfolio.
- **cover** (*float*) – cover per event for entire portfolio.

Returns

- **transfer_at_event** (*np.array*) – risk transfered per event
- **transfer_aai_agg** (*float*) – average risk within a period of 1/frequency_unit, transfered

residual_risk(*attachment*, *cover*)

Compute the residual risk after application of insurance attachment and cover to entire portfolio. This is the residual risk of the full portfolio summed over all events. For each event, the residual risk is obtained by subtracting the transfered risk from the trom the total risk per event. of the event.

Parameters

- **attachment** (*float*) – attachment per event for entire portfolio.
- **cover** (*float*) – cover per event for entire portfolio.

Returns

- **residual_at_event** (*np.array*) – residual risk per event
- **residual_aai_agg** (*float*) – average residual risk within a period of 1/frequency_unit

See also:

transfer_risk
compute the transfer risk per portfolio.

calc_risk_transfer(*attachment*, *cover*)

Compute traaditional risk transfer over impact. Returns new impact with risk transfer applied and the insurance layer resulting Impact metrics.

Parameters

- **attachment** (*float*) – (deductible)
- **cover** (*float*)

Return type

climada.engine.impact.Impact

impact_per_year(*all_years=True*, *year_range=None*)

Calculate yearly impact from impact data.

Note: the impact in a given year is summed over all events. Thus, the impact in a given year can be larger than the total affected exposure value.

Parameters

- **all_years** (*boolean, optional*) – return values for all years between first and last year with event, including years without any events. Default: True
- **year_range** (*tuple or list with integers, optional*) – start and end year

Returns

year_set – Key=year, value=Summed impact per year.

Return type

dict

impact_at_reg(*agg_regions=None*)

Aggregate impact on given aggregation regions. This method works only if Impact.imp_mat was stored during the impact calculation.

Parameters

agg_regions (*np.array, list (optional)*) – The length of the array must equal the number of centroids in exposures. It reports what macro-regions these centroids belong to. For example, asuming there are three centroids and agg_regions = ['A', 'A', 'B'] then impact of the first and second centroids will be assigned to region A, whereas impact from the third centroid will be assigned to area B. If no aggregation regions are passed, the method aggregates impact at the country (admin_0) level. Default is None.

Returns
Contains the aggregated data per event. Rows: Hazard events. Columns: Aggregation regions.

Return type
pd.DataFrame

calc_impact_year_set(*all_years=True*, *year_range=None*)

This function is deprecated, use Impact.impact_per_year instead.

local_exceedance_imp(*return_periods=(25, 50, 100, 250)*)

Compute exceedance impact map for given return periods. Requires attribute imp_mat.

Parameters
return_periods (*Any, optional*) – return periods to consider Dafault is (25, 50, 100, 250)

Return type
np.array

calc_freq_curve(*return_per=None*)

Compute impact exceedance frequency curve.

Parameters
return_per (*np.array, optional*) – return periods where to compute the exceedance impact. Use impact's frequencies if not provided

Return type
ImpactFreqCurve

plot_scatter_eai_exposure(*mask=None*, *ignore_zero=False*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *axis=None*, *adapt_fontsize=True*, ***kwargs*)

Plot scatter expected impact within a period of 1/frequency_unit of each exposure.

Parameters

- **mask** (*np.array, optional*) – mask to apply to eai_exp plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 1.0.
- **extend** (*str*) – optional extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **axis** (*matplotlib.axes.Axes, optional*) – axis to use
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **kwargs** (*dict, optional*) – arguments for hexbin matplotlib function

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_hexbin_eai_exposure(*mask=None*, *ignore_zero=False*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *axis=None*, *adapt_fontsize=True*, ***kwargs*)

Plot hexbin expected impact within a period of 1/frequency_unit of each exposure.

Parameters

- **mask** (*np.array, optional*) – mask to apply to eai_exp plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 1.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **axis** (*matplotlib.axes.Axes, optional*) – axis to use
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default: True
- **kwargs** (*dict, optional*) – arguments for hexbin matplotlib function

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_raster_eai_exposure(*res=None*, *raster_res=None*, *save_tiff=None*, *raster_f=<function Impact.<lambda>>*, *label='value (log10)'*, *axis=None*, *adapt_fontsize=True*, ***kwargs*)

Plot raster expected impact within a period of 1/frequency_unit of each exposure.

Parameters

- **res** (*float, optional*) – resolution of current data in units of latitude and longitude, approximated if not provided.
- **raster_res** (*float, optional*) – desired resolution of the raster
- **save_tiff** (*str, optional*) – file name to save the raster in tiff format, if provided
- **raster_f** (*lambda function*) – transformation to use to data. Default: log10 adding 1.
- **label** (*str*) – colorbar label
- **axis** (*matplotlib.axes.Axes, optional*) – axis to use
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **kwargs** (*dict, optional*) – arguments for imshow matplotlib function

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_basemap_eai_exposure(*mask=None*, *ignore_zero=False*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *zoom=10*, *url={'attribution': '(C) OpenStreetMap contributors (C) CARTO', 'html_attribution': '© OpenStreetMap contributors © CARTO', 'max_zoom': 20, 'name': 'CartoDB.Positron', 'subdomains': 'abcd', 'url': 'https://{s}.basemaps.cartocdn.com/{variant}/{z}/{x}/{y}{r}.png', 'variant': 'light_all'}*, *axis=None*, ***kwargs*)

Plot basemap expected impact of each exposure within a period of 1/frequency_unit.

Parameters

- **mask** (*np.array, optional*) – mask to apply to eai_exp plotted.

- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **zoom** (*int, optional*) – zoom coefficient used in the satellite image
- **url** (*str, optional*) – image source, default: ctx.providers.CartoDB.Positron
- **axis** (*matplotlib.axes.Axes, optional*) – axis to use
- **kwargs** (*dict, optional*) – arguments for scatter matplotlib function, e.g. cmap='Greys'. Default: 'Wistia'

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_hexbin_impact_exposure(*event_id=1*, *mask=None*, *ignore_zero=False*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *axis=None*, *adapt_fontsize=True*, ***kwargs*)

Plot hexbin impact of an event at each exposure. Requires attribute imp_mat.

Parameters

- **event_id** (*int, optional*) – id of the event for which to plot the impact. Default: 1.
- **mask** (*np.array, optional*) – mask to apply to impact plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 1.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **axis** (*matplotlib.axes.Axes*) – optional axis to use
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **kwargs** (*dict, optional*) – arguments for hexbin matplotlib function

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_basemap_impact_exposure(*event_id=1*, *mask=None*, *ignore_zero=False*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *zoom=10*, *url={'attribution': '(C) OpenStreetMap contributors (C) CARTO', 'html_attribution': '© OpenStreetMap contributors © CARTO', 'max_zoom': 20, 'name': 'CartoDB.Positron', 'subdomains': 'abcd', 'url': 'https://{s}.basemaps.cartocdn.com/{variant}/{z}/{x}/{y}{r}.png', 'variant': 'light_all'}*, *axis=None*, ***kwargs*)

Plot basemap impact of an event at each exposure. Requires attribute imp_mat.

Parameters

- **event_id** (*int, optional*) – id of the event for which to plot the impact. Default: 1.
- **mask** (*np.array, optional*) – mask to apply to impact plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **zoom** (*int, optional*) – zoom coefficient used in the satellite image
- **url** (*str, optional*) – image source, default: ctx.providers.CartoDB.Positron
- **axis** (*matplotlib.axes.Axes, optional*) – axis to use
- **kwargs** (*dict, optional*) – arguments for scatter matplotlib function, e.g. cmap='Greys'. Default: 'Wistia'

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_rp_imp(*return_periods=(25, 50, 100, 250)*, *log10_scale=True*, *smooth=True*, *axis=None*, ***kwargs*)

Compute and plot exceedance impact maps for different return periods. Calls local_exceedance_imp.

Parameters

- **return_periods** (*tuple of int, optional*) – return periods to consider. Default: (25, 50, 100, 250)
- **log10_scale** (*boolean, optional*) – plot impact as log10(impact). Default: True
- **smooth** (*bool, optional*) – smooth plot to plot.RESOLUTIONxplot.RESOLUTION. Default: True
- **kwargs** (*dict, optional*) – arguments for pcolormesh matplotlib function used in event plots

Returns

- **axis** (*matplotlib.axes.Axes*)
- **imp_stats** (*np.array*) – return_periods.size x num_centroids

write_csv(*file_name*)

Write data into csv file. imp_mat is not saved.

Parameters
file_name (*str*) – absolute path of the file

write_excel(*file_name*)

Write data into Excel file. imp_mat is not saved.

Parameters
file_name (*str*) – absolute path of the file

write_hdf5(*file_path: str | Path*, *dense_imp_mat: bool = False*)

Write the data stored in this object into an H5 file.

Try to write all attributes of this class into H5 datasets or attributes. By default, any iterable will be stored in a dataset and any string or scalar will be stored in an attribute. Dictionaries will be stored as groups, with the previous rules being applied recursively to their values.

The impact matrix can be stored in a sparse or dense format.

Notes

This writer does not support attributes with variable types. Please make sure that `event_name` is a list of equally-typed values, e.g., all `str`.

Parameters

- **file_path** (*str or Path*) – File path to write data into. The enclosing directory must exist.
- **dense_imp_mat** (*bool*) – If `True`, write the impact matrix as dense matrix that can be more easily interpreted by common H5 file readers but takes up (vastly) more space. Defaults to `False`.

write_sparse_csr(*file_name*)

Write imp_mat matrix in numpy's npz format.

static read_sparse_csr(*file_name*)

Read imp_mat matrix from numpy's npz format.

Parameters
file_name (*str*)

Return type
sparse.csr_matrix

classmethod from_csv(*file_name*)

Read csv file containing impact data generated by write_csv.

Parameters
file_name (*str*) – absolute path of the file

Returns
imp – Impact from csv file

Return type
climada.engine.impact.Impact

read_csv(**args*, ***kwargs*)

This function is deprecated, use Impact.from_csv instead.

classmethod from_excel(*file_name*)

Read excel file containing impact data generated by write_excel.

Parameters
file_name (*str*) – absolute path of the file

Returns
imp – Impact from excel file

Return type
climada.engine.impact.Impact

read_excel(**args*, ***kwargs*)

This function is deprecated, use Impact.from_excel instead.

classmethod from_hdf5(*file_path: str | Path*)

Create an impact object from an H5 file.

This assumes a specific layout of the file. If values are not found in the expected places, they will be set to the default values for an `Impact` object.

The following H5 file structure is assumed (H5 groups are terminated with `/`, attributes are denoted by `.attrs/`):

```
file.h5
├─ at_event
├─ coord_exp
├─ eai_exp
├─ event_id
├─ event_name
├─ frequency
├─ imp_mat
├─ .attrs/
│  ├─ aai_agg
│  ├─ crs
│  ├─ frequency_unit
│  ├─ haz_type
│  ├─ tot_value
│  ├─ unit
```

As per the *climada.engine.impact.Impact.__init__()*, any of these entries is optional. If it is not found, the default value will be used when constructing the Impact.

The impact matrix `imp_mat` can either be an H5 dataset, in which case it is interpreted as dense representation of the matrix, or an H5 group, in which case the group is expected to contain the following data for instantiating a scipy.sparse.csr_matrix:

```
imp_mat/
├─ data
├─ indices
├─ indptr
├─ .attrs/
│  ├─ shape
```

Parameters
file_path (*str or Path*) – The file path of the file to read.

Returns
imp – Impact with data from the given file

Return type
Impact

static video_direct_impact(*exp*, *impf_set*, *haz_list*, *file_name=''*, *writer=<matplotlib.animation.PillowWriter object>*, *imp_thresh=0*, *args_exp=None*, *args_imp=None*, *ignore_zero=False*, *pop_name=False*)

Computes and generates video of accumulated impact per input events over exposure.

Parameters

- **exp** (*climada.entity.Exposures*) – exposures instance, constant during all video
- **impf_set** (*climada.entity.ImpactFuncSet*) – impact functions
- **haz_list** (*(list(Hazard))*) – every Hazard contains an event; all hazards use the same centroids
- **file_name** (*str, optional*) – file name to save video, if provided
- **writer** (*matplotlib.animation.*, optional*) – video writer. Default: pillow with bitrate=500
- **imp_thresh** (*float, optional*) – represent damages greater than threshold. Default: 0

- **args_exp** (*dict, optional*) – arguments for scatter (points) or hexbin (raster) matplotlib function used in exposures
- **args_imp** (*dict, optional*) – arguments for scatter (points) or hexbin (raster) matplotlib function used in impact
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places The default is False.

Return type
list of *Impact*

select (*event_ids=None*, *event_names=None*, *dates=None*, *coord_exp=None*, *reset_frequency=False*)

Select a subset of events and/or exposure points from the impact. If multiple input variables are not None, it returns all the impacts matching at least one of the conditions.

Notes

the frequencies are NOT adjusted. Method to adjust frequencies

and obtain correct eai_exp:
1- Select subset of impact according to your choice imp = impact.select(…) 2- Adjust manually the frequency of the subset of impact imp.frequency = […] 3- Use select without arguments to select all events and recompute the eai_exp with the updated frequencies. imp = imp.select()

Parameters

- **event_ids** (*list of int, optional*) – Selection of events by their id. The default is None.
- **event_names** (*list of str, optional*) – Selection of events by their name. The default is None.
- **dates** (*tuple, optional*) – (start-date, end-date), events are selected if they are >= than start-date and <= than end-date. Dates in same format as impact.date (ordinal format of datetime library) The default is None.
- **coord_exp** (*np.array, optional*) – Selection of exposures coordinates [lat, lon] (in degrees) The default is None.
- **reset_frequency** (*bool, optional*) – Change frequency of events proportional to difference between first and last year (old and new). Assumes annual frequency values. Default: False.

Raises
`ValueError` – If the impact matrix is missing, the eai_exp and aai_agg cannot be updated for a selection of events and/or exposures.

Returns
imp – A new impact object with a selection of events and/or exposures

Return type
climada.engine.impact.Impact

classmethod concat (*imp_list: Iterable*, *reset_event_ids: bool = False*)

Concatenate impact objects with the same exposure

This function is useful if, e.g. different impact functions have to be applied for different seasons (e.g. for agricultural impacts).

It checks if the exposures of the passed impact objects are identical and then

- concatenates the attributes `event_id`, `event_name`, `date`, `frequency`, `imp_mat`, `at_event`,
- sums up the values of attributes `eai_exp`, `aai_exp`
- and takes the following attributes from the first impact object in the passed impact list: `coord_exp`, `crs`, `unit`, `tot_value`, `frequency_unit`, `haz_type`

If event ids are not unique among the passed impact objects an error is raised. In this case, the user can set `reset_event_ids=True` to create unique event ids for the concatenated impact.

If all impact matrices of the impacts in `imp_list` are empty, the impact matrix of the concatenated impact is also empty.

Parameters

- **imp_list** (*Iterable of climada.engine.impact.Impact*) – Iterable of Impact objects to concatenate
- **reset_event_ids** (*boolean, optional*) – Reset event ids of the concatenated impact object

Returns
impact – New impact object which is a concatenation of all impacts

Return type
climada.engine.impact.Impact

Notes

- Concatenation of impacts with different exposure (e.g. different countries) could also be implemented here in the future.

match_centroids(*hazard*, *distance='euclidean'*, *threshold=100*)

Finds the closest hazard centroid for each impact coordinate. Creates a temporary GeoDataFrame and uses `u_coord.match_centroids()`. See there for details and parameters

Parameters

- **hazard** (*Hazard*) – Hazard to match (with raster or vector centroids).
- **distance** (*str, optional*) – Distance to use in case of vector centroids. Possible values are "euclidean", "haversine" and "approx". Default: "euclidean"
- **threshold** (*float*) – If the distance (in km) to the nearest neighbor exceeds *threshold*, the index *-1* is assigned. Set *threshold* to 0, to disable nearest neighbor matching. Default: 100 (km)

Returns
array of closest Hazard centroids, aligned with the Impact's *coord_exp* array

Return type
np.array

5.1.6 climada.engine.impact_calc module

class climada.engine.impact_calc.**ImpactCalc**(*exposures*, *impfset*, *hazard*)

Bases: object

Class to compute impacts from exposures, impact function set and hazard

__init__(*exposures*, *impfset*, *hazard*)

ImpactCalc constructor

The dimension of the imp_mat variable must be compatible with the exposures and hazard objects.

Parameters

- **exposures** (*climada.entity.Exposures*) – exposures used to compute impacts
- **impf_set** (*climada.entity.ImpactFuncSet*) – impact functions set used to compute impacts
- **hazard** (*climada.Hazard*) – hazard used to compute impacts

property n_exp_pnt

Number of exposure points (rows in gdf)

property n_events

Number of hazard events (size of event_id array)

impact(*save_mat=True*, *assign_centroids=True*, *ignore_cover=False*, *ignore_deductible=False*)

Compute the impact of a hazard on exposures.

Parameters

- **save_mat** (*bool, optional*) – if true, save the total impact matrix (events x exposures) Default: True
- **assign_centroids** (*bool, optional*) – indicates whether centroids are assigned to the self.exposures object. Centroids assignment is an expensive operation; set this to False to save computation time if the hazards' centroids are already assigned to the exposures object. Default: True
- **ignore_cover** (*bool, optional*) – if set to True, the column 'cover' of the exposures GeoDataFrame, if present, is ignored and the impact it not capped by the values in this column. Default: False
- **ignore_deductible** (*bool, opotional*) – if set to True, the column 'deductible' of the exposures GeoDataFrame, if present, is ignored and the impact it not reduced through values in this column. Default: False

Examples

```
>>> haz = Hazard.from_mat(HAZ_DEMO_MAT)  # Set hazard
>>> impfset = ImpactFuncSet.from_excel(ENT_TEMPLATE_XLS)
>>> exp = Exposures(pd.read_excel(ENT_TEMPLATE_XLS))
>>> impcalc = ImpactCal(exp, impfset, haz)
>>> imp = impcalc.impact(insured=True)
>>> imp.aai_agg
```

See also:

apply_deductible_to_mat

apply deductible to impact matrix

apply_cover_to_mat
apply cover to impact matrix

minimal_exp_gdf (*impf_col*, *assign_centroids*, *ignore_cover*, *ignore_deductible*)

Get minimal exposures geodataframe for impact computation

Parameters

- **exposures** (*climada.entity.Exposures*)
- **hazard** (*climada.Hazard*)
- **impf_col** (*str*) – Name of the impact function column in exposures.gdf
- **assign_centroids** (*bool*) – Indicates whether centroids are re-assigned to the self.exposures object or kept from previous impact calculation with a hazard of the same hazard type. Centroids assignment is an expensive operation; set this to `False` to save computation time if the centroids have not changed since the last impact calculation.
- **include_cover** (*bool*) – if set to True, the column 'cover' of the exposures GeoDataFrame is excluded from the returned GeoDataFrame, otherwise it is included if present.
- **include_deductible** (*bool*) – if set to True, the column 'deductible' of the exposures GeoDataFrame is excluded from the returned GeoDataFrame, otherwise it is included if present.

imp_mat_gen (*exp_gdf*, *impf_col*)

Generator of impact sub-matrices and correspoding exposures indices

The exposures gdf is decomposed into chunks that fit into the max defined memory size. For each chunk, the impact matrix is computed and returned, together with the corresponding exposures points index.

Parameters

- **exp_gdf** (*GeoDataFrame*) – Geodataframe of the exposures with columns required for impact computation.
- **impf_col** (*str*) – name of the desired impact column in the exposures.

Raises

`ValueError` – if the hazard is larger than the memory limit

Yields

scipy.sparse.crs_matrix, np.ndarray – impact matrix and corresponding exposures indices for each chunk.

insured_mat_gen (*imp_mat_gen*, *exp_gdf*, *impf_col*)

Generator of insured impact sub-matrices (with applied cover and deductible) and corresponding exposures indices

This generator takes a 'regular' impact matrix generator and applies cover and deductible onto the impacts. It yields the same sub-matrices as the original generator.

Deductible and cover are taken from the dataframe stored in *exposures.gdf*.

Parameters

- **imp_mat_gen** (*generator of tuples (sparse.csr_matrix, np.array)*) – The generator for creating the impact matrix. It returns a part of the full matrix and the associated exposure indices.
- **exp_gdf** (*GeoDataFrame*) – Geodataframe of the exposures with columns required for impact computation.
- **impf_col** (*str*) – Name of the column in 'exp_gdf' indicating the impact function (id)

Yields

- **mat** (*scipy.sparse.csr_matrix*) – Impact sub-matrix (with applied cover and deductible) with size (n_events, len(exp_idx))
- **exp_idx** (*np.array*) – Exposure indices for impacts in mat

impact_matrix(*exp_values*, *cent_idx*, *impf*)

Compute the impact matrix for given exposure values, assigned centroids, a hazard, and one impact function.

Parameters

- **exp_values** (*np.array*) – Exposure values
- **cent_idx** (*np.array*) – Hazard centroids assigned to each exposure location
- **hazard** (*climada.Hazard*) – Hazard object
- **impf** (*climada.entity.ImpactFunc*) – one impactfunction comon to all exposure elements in exp_gdf

Returns

Impact per event (rows) per exposure point (columns)

Return type

scipy.sparse.csr_matrix

stitch_impact_matrix(*imp_mat_gen*)

Make an impact matrix from an impact sub-matrix generator

stitch_risk_metrics(*imp_mat_gen*)

Compute the impact metrics from an impact sub-matrix generator

This method is used to compute the risk metrics if the user decided not to store the full impact matrix.

Parameters

imp_mat_gen (*generator of tuples (sparse.csr_matrix, np.array)*) – The generator for creating the impact matrix. It returns a part of the full matrix and the associated exposure indices.

Returns

- **at_event** (*np.array*) – Accumulated damage for each event
- **eai_exp** (*np.array*) – Expected impact within a period of 1/frequency_unit for each exposure point
- **aai_agg** (*float*) – Average impact within a period of 1/frequency_unit aggregated

static apply_deductible_to_mat(*mat*, *deductible*, *hazard*, *cent_idx*, *impf*)

Apply a deductible per exposure point to an impact matrix at given centroid points for given impact function.

All exposure points must have the same impact function. For different impact functions apply use this method repeatedly on the same impact matrix.

Parameters

- **imp_mat** (*scipy.sparse.csr_matrix*) – impact matrix (events x exposure points)
- **deductible** (*np.array()*) – deductible for each exposure point
- **hazard** (*climada.Hazard*) – hazard used to compute the imp_mat
- **cent_idx** (*np.array()*) – index of centroids associated with each exposure point
- **impf** (*climada.entity.ImpactFunc*) – impact function associated with the exposure points

Returns
imp_mat – impact matrix with applied deductible

Return type
scipy.sparse.csr_matrix

static apply_cover_to_mat(*mat*, *cover*)

Apply cover to impact matrix.

The impact data is clipped to the range [0, cover]. The cover is defined per exposure point.

Parameters

- **imp_mat** (*scipy.sparse.csr_matrix*) – impact matrix
- **cover** (*np.array()*) – cover per exposures point (columns of imp_mat)

Returns
imp_mat – impact matrix with applied cover

Return type
scipy.sparse.csr_matrix

static eai_exp_from_mat(*mat*, *freq*)

Compute impact for each exposures from the total impact matrix

Parameters

- **imp_mat** (*sparse.csr_matrix*) – matrix num_events x num_exp with impacts.
- **frequency** (*np.array*) – frequency of events within a period of 1/frequency_unit

Returns
eai_exp – expected impact within a period of 1/frequency_unit for each exposure

Return type
np.array

static at_event_from_mat(*mat*)

Compute impact for each hazard event from the total impact matrix :Parameters: **imp_mat** (*sparse.csr_matrix*) – matrix num_events x num_exp with impacts.

Returns
at_event – impact for each hazard event

Return type
np.array

static aai_agg_from_eai_exp(*eai_exp*)

Aggregate impact.eai_exp

Parameters
eai_exp (*np.array*) – expected impact within a period of 1/frequency_unit for each exposure point

Returns
average aggregated impact within a period of 1/frequency_unit

Return type
float

classmethod risk_metrics(*mat*, *freq*)

Compute risk metricss eai_exp, at_event, aai_agg for an impact matrix and a frequency vector.

Parameters

- **mat** (*sparse.csr_matrix*) – matrix num_events x num_exp with impacts.
- **freq** (*np.array*) – array with the frequency per event

Returns

- **eai_exp** (*np.array*) – expected impact within a period of 1/frequency_unit at each exposure point
- **at_event** (*np.array()*) – total impact for each event
- **aai_agg** (*float*) – average impact within a period of 1/frequency_unit aggregated over all exposure points

5.1.7 climada.engine.impact_data module

`climada.engine.impact_data.`**`assign_hazard_to_emdat`**(*certainty_level*, *intensity_path_haz*, *names_path_haz*, *reg_id_path_haz*, *date_path_haz*, *emdat_data*, *start_time*, *end_time*, *keep_checks=False*)

assign_hazard_to_emdat: link EMdat event to hazard

Parameters

- **certainty_level** (*str*) – 'high' or 'low'
- **intensity_path_haz** (*sparse matrix*) – with hazards as rows and grid points as cols, values only at location with impacts
- **names_path_haz** (*str*) – identifier for each hazard (i.e. IBtracID) (rows of the matrix)
- **reg_id_path_haz** (*str*) – ISO country ID of each grid point (cols of the matrix)
- **date_path_haz** (*str*) – start date of each hazard (rows of the matrix)
- **emdat_data** (*pd.DataFrame*) – dataframe with EMdat data
- **start_time** (*str*) – start date of events to be assigned 'yyyy-mm-dd'
- **end_time** (*str*) – end date of events to be assigned 'yyyy-mm-dd'
- **keep_checks** (*bool, optional*)

Return type

pd.dataframe with EMdat entries linked to a hazard

`climada.engine.impact_data.`**`hit_country_per_hazard`**(*intensity_path*, *names_path*, *reg_id_path*, *date_path*)

hit_country_per_hazard: create list of hit countries from hazard set

Parameters

- **intensity_path** (*str*) – Path to file containing sparse matrix with hazards as rows and grid points as cols, values only at location with impacts
- **names_path** (*str*) – Path to file with identifier for each hazard (i.e. IBtracID) (rows of the matrix)
- **reg_id_path** (*str*) – Path to file with ISO country ID of each grid point (cols of the matrix)
- **date_path** (*str*) – Path to file with start date of each hazard (rows of the matrix)

Return type
pd.DataFrame with all hit countries per hazard

`climada.engine.impact_data.`**`create_lookup`**(*emdat_data*, *start*, *end*, *disaster_subtype='Tropical cyclone'*)

create_lookup: prepare a lookup table of EMdat events to which hazards can be assigned

Parameters

- **emdat_data** (*pd.DataFrame*) – with EMdat data
- **start** (*str*) – start date of events to be assigned 'yyyy-mm-dd'
- **end** (*str*) – end date of events to be assigned 'yyyy-mm-dd'
- **disaster_subtype** (*str*) – EMdat disaster subtype

Return type
pd.DataFrame

`climada.engine.impact_data.`**`emdat_possible_hit`**(*lookup*, *hit_countries*, *delta_t*)

relate EM disaster to hazard using hit countries and time

Parameters

- **lookup** (*pd.DataFrame*) – to relate EMdatID to hazard
- **delta_t** – max time difference of start of EMdat event and hazard
- **hit_countries**

Return type
list with possible hits

`climada.engine.impact_data.`**`match_em_id`**(*lookup*, *poss_hit*)

function to check if EM_ID has been assigned already and combine possible hits

Parameters

- **lookup** (*pd.dataframe*) – to relate EMdatID to hazard
- **poss_hit** (*list*) – with possible hits

Returns
with all possible hits per EMdat ID

Return type
list

`climada.engine.impact_data.`**`assign_track_to_em`**(*lookup*, *possible_tracks_1*, *possible_tracks_2*, *level*)

function to assign a hazard to an EMdat event to get some confidene into the procedure, hazards get only assigned if there is no other hazard occuring at a bigger time interval in that country Thus a track of possible_tracks_1 gets only assigned if there are no other tracks in possible_tracks_2. The confidence can be expressed with a certainty level

Parameters

- **lookup** (*pd.DataFrame*) – to relate EMdatID to hazard
- **possible_tracks_1** (*list*) – list of possible hits with smaller time horizon
- **possible_tracks_2** (*list*) – list of possible hits with larger time horizon
- **level** (*int*) – level of confidence

Returns
lookup with assigend tracks and possible hits

Return type
pd.DataFrame

`climada.engine.impact_data.`**`check_assigned_track`**(*lookup*, *checkset*)

compare lookup with assigned tracks to a set with checked sets

Parameters

- **lookup** (*pd.DataFrame*) – dataframe to relate EMdatID to hazard
- **checkset** (*pd.DataFrame*) – dataframe with already checked hazards

Return type
error scores

`climada.engine.impact_data.`**`clean_emdat_df`**(*emdat_file*, *countries=None*, *hazard=None*, *year_range=None*, *target_version=None*)

Get a clean and standardized DataFrame from EM-DAT-CSV-file (1) load EM-DAT data from CSV to DataFrame and remove header/footer, (2) handle version, clean up, and add columns, and (3) filter by country, hazard type and year range (if any given)

Parameters

- **emdat_file** (*str, Path, or DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV
- **countries** (*list of str*) – country ISO3-codes or names, e.g. ['JAM', 'CUB']. countries=None for all countries (default)
- **hazard** (*list or str*) – List of Disaster (sub-)type accordung EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.
- **year_range** (*list or tuple*) – Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)
- **target_version** (*int*) – required EM-DAT data format version (i.e. year of download), changes naming of columns/variables, default: newest available version in `VARNAMES_EMDAT` that matches the given emdat_file

Returns
df_data – DataFrame containing cleaned and filtered EM-DAT impact data

Return type
pd.DataFrame

`climada.engine.impact_data.`**`emdat_countries_by_hazard`**(*emdat_file_csv*, *hazard=None*, *year_range=None*)

return list of all countries exposed to a chosen hazard type from EMDAT data as CSV.

Parameters

- **emdat_file** (*str, Path, or DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV
- **hazard** (*list or str*) – List of Disaster (sub-)type accordung EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal

Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.

- **year_range** (*list or tuple*) – Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)

Returns

- **countries_iso3a** (*list*) – List of ISO3-codes of countries impacted by the disaster (sub-)types
- **countries_names** (*list*) – List of names of countries impacted by the disaster (sub-)types

`climada.engine.impact_data.`**`scale_impact2refyear`**(*impact_values*, *year_values*, *iso3a_values*, *reference_year=None*)

Scale give impact values proportional to GDP to the according value in a reference year (for normalization of monetary values)

Parameters

- **impact_values** (*list or array*) – Impact values to be scaled.
- **year_values** (*list or array*) – Year of each impact (same length as impact_values)
- **iso3a_values** (*list or array*) – ISO3alpha code of country for each impact (same length as impact_values)
- **reference_year** (*int, optional*) – Impact is scaled proportional to GDP to the value of the reference year. No scaling for reference_year=None (default)

`climada.engine.impact_data.`**`emdat_impact_yearlysum`**(*emdat_file_csv*, *countries=None*, *hazard=None*, *year_range=None*, *reference_year=None*, *imp_str="Total Damages ('000 US$)"*, *version=None*)

function to load EM-DAT data and sum impact per year

Parameters

emdat_file_csv (*str or DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV

countries

[list of str] country ISO3-codes or names, e.g. ['JAM', 'CUB']. countries=None for all countries (default)

hazard

[list or str] List of Disaster (sub-)type accordung EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.

year_range

[list or tuple] Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)

version

[int, optional] required EM-DAT data format version (i.e. year of download), changes naming of columns/variables, default: newest available version in `VARNAMES_EMDAT`

Returns

out – DataFrame with summed impact and scaled impact per year and country.

Return type

pd.DataFrame

`climada.engine.impact_data.`**`emdat_impact_event`**(*emdat_file_csv*, *countries=None*, *hazard=None*, *year_range=None*, *reference_year=None*, *imp_str="Total Damages ('000 US$)"*, *version=None*)

function to load EM-DAT data return impact per event

Parameters
emdat_file_csv (*str or DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV

countries
[list of str] country ISO3-codes or names, e.g. ['JAM', 'CUB']. default: countries=None for all countries

hazard
[list or str] List of Disaster (sub-)type accordung EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.

year_range
[list or tuple] Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)

reference_year
[int reference year of exposures. Impact is scaled] proportional to GDP to the value of the reference year. Default: No scaling for 0

imp_str
[str] Column name of impact metric in EMDAT CSV, default = "Total Damages ('000 US$)"

version
[int, optional] EM-DAT version to take variable/column names from, default: newest available version in `VARNAMES_EMDAT`

Returns
out – EMDAT DataFrame with new columns "year", "region_id", and "impact" and +impact_scaled" total impact per event with same unit as chosen impact, but multiplied by 1000 if impact is given as 1000 US$ (e.g. imp_str="Total Damages ('000 US$) scaled").

Return type
pd.DataFrame

`climada.engine.impact_data.`**`emdat_to_impact`**(*emdat_file_csv*, *hazard_type_climada*, *year_range=None*, *countries=None*, *hazard_type_emdat=None*, *reference_year=None*, *imp_str='Total Damages'*)

function to load EM-DAT data return impact per event

Parameters

- **emdat_file_csv** (*str or pd.DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV
- **countries** (*list of str*) – country ISO3-codes or names, e.g. ['JAM', 'CUB']. default: countries=None for all countries
- **hazard_type_climada** (*str*) – CLIMADA hazard type abbreviations, e.g. TC, BF, etc.

- **hazard_type_emdat** (*list or str*) – List of Disaster (sub-)type accordung EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.
- **year_range** (*list or tuple*) – Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)
- **reference_year** (*int reference year of exposures. Impact is scaled*) – proportional to GDP to the value of the reference year. Default: No scaling for 0
- **imp_str** (*str*) – Column name of impact metric in EMDAT CSV, default = "Total Damages ('000 US$)"

Returns

- **impact_instance** (*climada.engine.Impact*) – impact object of same format as output from CLIMADA impact computation. Values scaled with GDP to reference_year if reference_year is given. i.e. current US$ for imp_str="Total Damages ('000 US$) scaled" (factor 1000 is applied) impact_instance.eai_exp holds expected impact for each country (within 1/frequency_unit). impact_instance.coord_exp holds rough central coordinates for each country.
- **countries** (*list of str*) – ISO3-codes of countries in same order as in impact_instance.eai_exp

5.2 climada.entity package

5.2.1 climada.entity.disc_rates package

climada.entity.disc_rates.base module

class `climada.entity.disc_rates.base.`**`DiscRates`**(*years: ndarray | None = None*, *rates: ndarray | None = None*)

Bases: `object`

Defines discount rates and basic methods. Loads from files with format defined in FILE_EXT.

`years`

list of years

Type

np.array

`rates`

list of discount rates for each year (between 0 and 1)

Type

np.array

`__init__`(*years: ndarray | None = None*, *rates: ndarray | None = None*)

Fill discount rates with values and check consistency data

Parameters

- **years** (*numpy.ndarray(int)*) – Array of years. Default is numpy.array([]).
- **rates** (*numpy.ndarray(float)*) – Discount rates for each year in years. Default is numpy.array([]). Note: rates given in float, e.g., to set 1% rate use 0.01

clear()

Reinitialize attributes.

check()

Check attributes consistency.

Raises

ValueError –

select(*year_range*)

Select discount rates in given years.

Parameters

- **year_range** (*np.array(int)*) – continuous sequence of selected years.
- **Returns** (*climada.entity.DiscRates*) – The selected discrates in the year_range

append(*disc_rates*)

Check and append discount rates to current DiscRates. Overwrite discount rate if same year.

Parameters

disc_rates (*climada.entity.DiscRates*) – DiscRates instance to append

Raises

ValueError –

net_present_value(*ini_year*, *end_year*, *val_years*)

Compute net present value between present year and future year.

Parameters

- **ini_year** (*float*) – initial year
- **end_year** (*float*) – end year
- **val_years** (*np.array*) – cash flow at each year btw ini_year and end_year (both included)

Returns

net_present_value – net present value between present year and future year.

Return type

float

plot(*axis=None*, *figsize=(6, 8)*, ***kwargs*)

Plot discount rates per year.

Parameters

- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*tuple(int, int), optional*) – size of the figure. The default is (6,8)
- **kwargs** (*optional*) – keyword arguments passed to plotting function axis.plot

Returns

axis – axis handles of the plot

Return type

matplotlib.axes._subplots.AxesSubplot

classmethod from_mat(*file_name*, *var_names=None*)

Read MATLAB file generated with previous MATLAB CLIMADA version.

Parameters

- **file_name** (*str*) – filename including path and extension
- **description** (*str, optional*) – description of the data. The default is ''
- **var_names** (*dict, optional*) – name of the variables in the file. Default:

```
>>> DEF_VAR_MAT = {
...     'sup_field_name': 'entity',
...     'field_name': 'discount',
...     'var_name': {
...         'year': 'year',
...         'disc': 'discount_rate',
...     }
... }
```

Returns
The disc rates from matlab

Return type
climada.entity.DiscRates

read_mat(**args*, ***kwargs*)

This function is deprecated, use `DiscRates.from_mat` instead.

classmethod from_excel(*file_name*, *var_names=None*)

Read excel file following template and store variables.

Parameters

- **file_name** (*str*) – filename including path and extension
- **description** (*str, optional*) – description of the data. The default is ''
- **var_names** (*dict, optional*) – name of the variables in the file. The Default is

```
>>> DEF_VAR_EXCEL = {
...     'sheet_name': 'discount',
...     'col_name': {
...         'year': 'year',
...         'disc': 'discount_rate',
...     }
... }
```

Returns
The disc rates from excel

Return type
climada.entity.DiscRates

read_excel(**args*, ***kwargs*)

This function is deprecated, use DiscRates.from_excel instead.

write_excel(*file_name*, *var_names=None*)

Write excel file following template.

Parameters

- **file_name** (*str*) – filename including path and extension
- **var_names** (*dict, optional*) – name of the variables in the file. The Default is

```
>>> DEF_VAR_EXCEL = {
...     'sheet_name': 'discount',
...     'col_name': {
...         'year': 'year',
...         'disc': 'discount_rate',
...     }
... }
```

classmethod from_csv(*file_name*, *year_column='year'*, *disc_column='discount_rate'*, ***kwargs*)

Read DiscRate from a csv file following template and store variables.

Parameters

- **file_name** (*str*) – filename including path and extension
- **year_column** (*str, optional*) – name of the column that contains the years, Default: "year"
- **disc_column** (*str, optional*) – name of the column that contains the discount rates, Default: "discount_rate"
- ****kwargs** – any additional arguments, e.g., *sep*, *delimiter*, *head*, are forwarded to `pandas.read_csv`

Returns

The disc rates from the csv file

Return type

climada.entity.DiscRates

write_csv(*file_name*, *year_column='year'*, *disc_column='discount_rate'*, ***kwargs*)

Write DiscRate to a csv file following template and store variables.

Parameters

- **file_name** (*str*) – filename including path and extension
- **year_column** (*str, optional*) – name of the column that contains the years, Default: "year"
- **disc_column** (*str, optional*) – name of the column that contains the discount rates, Default: "discount_rate"
- ****kwargs** – any additional arguments, e.g., *sep*, *delimiter*, *head*, are forwarded to `pandas.read_csv`

5.2.2 climada.entity.exposures package

climada.entity.exposures.litpop package

climada.entity.exposures.litpop.gpw_population module

`climada.entity.exposures.litpop.gpw_population.`**load_gpw_pop_shape**(*geometry*, *reference_year*, *gpw_version*, *data_dir=PosixPath('/home/docs/clim*, *layer=0*, *verbose=True*)

Read gridded population data from TIFF and crop to given shape(s).

Note: A (free) NASA Earthdata login is necessary to download the data. Data can be downloaded e.g. for gpw_version=11 and year 2015 from https://sedac.ciesin.columbia.edu/downloads/data/gpw-v4/ gpw-v4-population-count-rev11/gpw-v4-population-count-rev11_2015_30_sec_tif.zip

Parameters

- **geometry** (*shape(s) to crop data to in degree lon/lat.*) – for example shapely.geometry.(Multi)Polygon or shapefile.Shape from polygon(s) defined in a (country) shapefile.
- **reference_year** (*int*) – target year for data extraction
- **gpw_version** (*int*) – Version number of GPW population data, i.e. 11 for v4.11. The default is CONFIG.exposures.litpop.gpw_population.gpw_version.int()
- **data_dir** (*Path, optional*) – Path to data directory holding GPW data folders. The default is SYSTEM_DIR.
- **layer** (*int, optional*) – relevant data layer in input TIFF file to return. The default is 0 and should not be changed without understanding the different data layers in the given TIFF file.
- **verbose** (*bool, optional*) – Enable verbose logging about the used GPW version and reference year. Default: True.

Returns

- **pop_data** (*2D numpy array*) – contains extracted population count data per grid point in shape first dimension is lat, second dimension is lon.
- **meta** (*dict*) – contains meta data per array, including "transform" with meta data on coordinates.
- **global_transform** (*Affine instance*) – contains six numbers, providing transform info for global GWP grid. global_transform is required for resampling on a globally consistent grid

`climada.entity.exposures.litpop.gpw_population.`**`get_gpw_file_path`**(*gpw_version*, *reference_year*, *data_dir=None*, *verbose=True*)

Check available GPW population data versions and year closest to *reference_year* and return full path to TIFF file.

Parameters

- **gpw_version** (*int (optional)*) – Version number of GPW population data, i.e. 11 for v4.11.
- **reference_year** (*int (optional)*) – Data year is selected as close to reference_year as possible. The default is 2020.
- **data_dir** (*pathlib.Path (optional)*) – Absolute path where files are stored. Default: SYSTEM_DIR
- **verbose** (*bool, optional*) – Enable verbose logging about the used GPW version and reference year. Default: True.

Raises

`FileExistsError` –

Returns

pathlib.Path

Return type

path to input file with population data

climada.entity.exposures.litpop.litpop module

climada.entity.exposures.litpop.litpop.**GPW_VERSION = 11**

Version of Gridded Population of the World (GPW) input data. Check for updates.

class climada.entity.exposures.litpop.litpop.**LitPop**(**args*, *meta=None*, *description=None*, *ref_year=2018*, *value_unit='USD'*, *crs=None*, ***kwargs*)

Bases: *Exposures*

Holds geopandas GeoDataFrame with metada and columns (pd.Series) defined in Attributes of Exposures class. LitPop exposure values are disaggregated proportional to a combination of nightlight intensity (NASA) and Gridded Population data (SEDAC). Total asset values can be produced capital, population count, GDP, or non-financial wealth.

Calling sequence example: country_names = ['CHE', 'Austria'] exp = LitPop.from_countries(country_names) exp.plot()

exponents

Defining powers (m, n) with which lit (nightlights) and pop (gpw) go into Lit**m * Pop**n. The default is (1,1).

Type

tuple of two integers, optional

fin_mode

Socio-economic value to be used as an asset base that is disaggregated. The default is 'pc'.

Type

str, optional

gpw_version

Version number of GPW population data, e.g. 11 for v4.11. The default is defined in GPW_VERSION.

Type

int, optional

set_countries(**args*, ***kwargs*)

This function is deprecated, use LitPop.from_countries instead.

classmethod from_countries(*countries*, *res_arcsec=30*, *exponents=(1, 1)*, *fin_mode='pc'*, *total_values=None*, *admin1_calc=False*, *reference_year=2018*, *gpw_version=11*, *data_dir=PosixPath('/home/docs/climada/data')*)

Init new LitPop exposure object for a list of countries (admin 0).

Sets attributes *ref_year*, *crs*, *value*, *geometry*, *meta*, *value_unit*, *exponents*,`fin_mode`, *gpw_version*, and *admin1_calc*.

Parameters

- **countries** (*list with str or int*) – list containing country identifiers: iso3alpha (e.g. 'JPN'), iso3num (e.g. 92) or name (e.g. 'Togo')
- **res_arcsec** (*float, optional*) – Horizontal resolution in arc-sec. The default is 30 arcsec, this corresponds to roughly 1 km.
- **exponents** (*tuple of two integers, optional*) – Defining power with which lit (nightlights) and pop (gpw) go into LitPop. To get nightlights^3 without population count: (3, 0). To use population count alone: (0, 1). Default: (1, 1)

- **fin_mode** (*str, optional*) – Socio-economic value to be used as an asset base that is disaggregated to the grid points within the country:
  - 'pc': produced capital (Source: World Bank), incl. manufactured or built assets such as machinery, equipment, and physical structures *pc* is in constant 2014 USD.
  - 'pop': population count (source: GPW, same as gridded population). The unit is 'people'.
  - 'gdp': gross-domestic product (Source: World Bank) [USD]
  - 'income_group': gdp multiplied by country's income group+1 [USD]. Income groups are 1 (low) to 4 (high income).
  - 'nfw': non-financial wealth (Source: Credit Suisse, of households only) [USD]
  - 'tw': total wealth (Source: Credit Suisse, of households only) [USD]
  - 'norm': normalized by country (no unit)
  - 'none': LitPop per pixel is returned unchanged (no unit)

  Default: 'pc'
- **total_values** (*list containing numerics, same length as countries, optional*) – Total values to be disaggregated to grid in each country. The default is None. If None, the total number is extracted from other sources depending on the value of fin_mode.
- **admin1_calc** (*boolean, optional*) – If True, distribute admin1-level GDP (if available). Default: False
- **reference_year** (*int, optional*) – Reference year. Default: CONFIG.exposures.def_ref_year.
- **gpw_version** (*int, optional*) – Version number of GPW population data. The default is GPW_VERSION
- **data_dir** (*Path, optional*) – redefines path to input data directory. The default is SYSTEM_DIR.

Raises
`ValueError` –

Returns
exp – LitPop instance with exposure for given countries

Return type
LitPop

set_nightlight_intensity(**args*, ***kwargs*)

This function is deprecated, use LitPop.from_nightlight_intensity instead.

classmethod from_nightlight_intensity(*countries=None*, *shape=None*, *res_arcsec=15*, *reference_year=2018*, *data_dir=PosixPath('/home/docs/climada/data')*)

Wrapper around *from_countries* / *from_shape*.

Initiate exposures instance with value equal to the original BlackMarble nightlight intensity resampled to the target resolution *res_arcsec*.

Provide either *countries* or *shape*.

Parameters

- **countries** (*list or str, optional*) – list containing country identifiers (name or iso3)

- **shape** (*Shape, Polygon or MultiPolygon, optional*) – geographical shape of target region, alternative to *countries*.
- **res_arcsec** (*int, optional*) – Resolution in arc seconds. The default is 15.
- **reference_year** (*int, optional*) – Reference year. The default is CONFIG.exposures.def_ref_year.
- **data_dir** (*Path, optional*) – data directory. The default is None.

Raises
`ValueError` –

Returns
exp – Exposure instance with values representing pure nightlight intensity from input nightlight data (BlackMarble)

Return type
LitPop

`set_population` (**args*, ***kwargs*)

This function is deprecated, use LitPop.from_population instead.

`classmethod from_population` (*countries=None*, *shape=None*, *res_arcsec=30*, *reference_year=2018*, *gpw_version=11*, *data_dir=PosixPath('/home/docs/climada/data')*)

Wrapper around *from_countries* / *from_shape*.

Initiate exposures instance with value equal to GPW population count. Provide either *countries* or *shape*.

Parameters

- **countries** (*list or str, optional*) – list containing country identifiers (name or iso3)
- **shape** (*Shape, Polygon or MultiPolygon, optional*) – geographical shape of target region, alternative to *countries*.
- **res_arcsec** (*int, optional*) – Resolution in arc seconds. The default is 30.
- **reference_year** (*int, optional*) – Reference year (closest available GPW data year is used) The default is CONFIG.exposures.def_ref_year.
- **gpw_version** (*int, optional*) – specify GPW data verison. The default is 11.
- **data_dir** (*Path, optional*) – data directory. The default is None. Either countries or shape is required.

Raises
`ValueError` –

Returns
exp – Exposure instance with values representing population count according to Gridded Population of the World (GPW) input data set.

Return type
LitPop

`set_custom_shape_from_countries` (**args*, ***kwargs*)

This function is deprecated, use LitPop.from_shape_and_countries instead.

`classmethod from_shape_and_countries` (*shape*, *countries*, *res_arcsec=30*, *exponents=(1, 1)*, *fin_mode='pc'*, *admin1_calc=False*, *reference_year=2018*, *gpw_version=11*, *data_dir=PosixPath('/home/docs/climada/data')*) create LitPop exposure for *country* and then crop to given shape.

Parameters

- **shape** (*shapely.geometry.Polygon, MultiPolygon, shapereader.Shape,*) – or GeoSeries or list containg either Polygons or Multipolygons. Geographical shape for which LitPop Exposure is to be initiated.
- **countries** (*list with str or int*) – list containing country identifiers: iso3alpha (e.g. 'JPN'), iso3num (e.g. 92) or name (e.g. 'Togo')
- **res_arcsec** (*float, optional*) – Horizontal resolution in arc-sec. The default is 30 arcsec, this corresponds to roughly 1 km.
- **exponents** (*tuple of two integers, optional*) – Defining power with which lit (nightlights) and pop (gpw) go into LitPop. Default: (1, 1)
- **fin_mode** (*str, optional*) – Socio-economic value to be used as an asset base that is disaggregated to the grid points within the country:
  - 'pc': produced capital (Source: World Bank), incl. manufactured or built assets such as machinery, equipment, and physical structures (pc is in constant 2014 USD)
  - 'pop': population count (source: GPW, same as gridded population). The unit is 'people'.
  - 'gdp': gross-domestic product (Source: World Bank) [USD]
  - 'income_group': gdp multiplied by country's income group+1 [USD] Income groups are 1 (low) to 4 (high income).
  - 'nfw': non-financial wealth (Source: Credit Suisse, of households only) [USD]
  - 'tw': total wealth (Source: Credit Suisse, of households only) [USD]
  - 'norm': normalized by country
  - 'none': LitPop per pixel is returned unchanged

  Default: 'pc'
- **admin1_calc** (*boolean, optional*) – If True, distribute admin1-level GDP (if available). Default: False
- **reference_year** (*int, optional*) – Reference year for data sources. Default: 2020
- **gpw_version** (*int, optional*) – Version number of GPW population data. The default is GPW_VERSION
- **data_dir** (*Path, optional*) – redefines path to input data directory. The default is SYSTEM_DIR.

Raises

`NotImplementedError` –

Returns

exp – The exposure LitPop within shape

Return type

LitPop

set_custom_shape(**args*, ***kwargs*)

This function is deprecated, use LitPop.from_shape instead.

classmethod from_shape(*shape*, *total_value*, *res_arcsec=30*, *exponents=(1, 1)*, *value_unit='USD'*, *region_id=None*, *reference_year=2018*, *gpw_version=11*, *data_dir=PosixPath('/home/docs/climada/data')*)

init LitPop exposure object for a custom shape. Requires user input regarding the total value to be disaggregated.

Sets attributes *ref_year*, *crs*, *value*, *geometry*, *meta*, *value_unit*, *exponents*,`fin_mode`, *gpw_version*, and *admin1_calc*.

This method can be used to initiated LitPop Exposure for sub-national regions such as states, districts, cantons, cities, … but shapes and total value need to be provided manually. If these required input parameters are not known / available, better initiate Exposure for entire country and extract shape afterwards.

Parameters

- **shape** (*shapely.geometry.Polygon or MultiPolygon or shapereader.Shape.*) – Geographical shape for which LitPop Exposure is to be initiated.
- **total_value** (*int, float or None type*) – Total value to be disaggregated to grid in shape. If None, no value is disaggregated.
- **res_arcsec** (*float, optional*) – Horizontal resolution in arc-sec. The default 30 arcsec corresponds to roughly 1 km.
- **exponents** (*tuple of two integers, optional*) – Defining power with which lit (nightlights) and pop (gpw) go into LitPop.
- **value_unit** (*str*) – Unit of exposure values. The default is USD.
- **region_id** (*int, optional*) – The numeric ISO 3166 region associated with the shape. If set to a value, this single value will be set for every coordinate in the `GeoDataFrame` of the resulting `LitPop` instance. If `None` (default), the region ID for every coordinate will be determined automatically (at a slight computational cost).
- **reference_year** (*int, optional*) – Reference year for data sources. Default: CONFIG.exposures.def_ref_year
- **gpw_version** (*int, optional*) – Version number of GPW population data. The default is set in CONFIG.
- **data_dir** (*Path, optional*) – redefines path to input data directory. The default is SYSTEM_DIR.

Raises

- **NotImplementedError** –
- **ValueError** –
- **TypeError** –

Returns

exp – The exposure LitPop within shape

Return type

LitPop

set_country(**args*, ***kwargs*)

This function is deprecated, use LitPop.from_countries instead.

climada.entity.exposures.litpop.litpop.**get_value_unit**(*fin_mode*)

get *value_unit* depending on *fin_mode*

Parameters
fin_mode (*Socio-economic value to be used as an asset base*)

Returns
value_unit

Return type
str

`climada.entity.exposures.litpop.litpop.`**`reproject_input_data`**(*data_array_list*, *meta_list*, *i_align=0*, *target_res_arcsec=None*, *global_origins=(-180.0, 89.99999999999991)*, *resampling=Resampling.bilinear*, *conserve=None*)

LitPop-sepcific wrapper around u_coord.align_raster_data.

Reprojects all arrays in data_arrays to a given resolution – all based on the population data grid.

Parameters

- **data_array_list** (*list or array of numpy arrays containing numbers*) – Data to be reprojected, i.e. list containing N (min. 1) 2D-arrays. The data with the reference grid used to align the global destination grid to should be first data_array_list[i_align], e.g., pop (GPW population data) for LitPop.
- **meta_list** (*list of dicts*) – meta data dictionaries of data arrays in same order as data_array_list. Required fields in each dict are 'dtype,', 'width', 'height', 'crs', 'transform'. Example:

```
>>> {
...     'driver': 'GTiff',
...     'dtype': 'float32',
...     'nodata': 0,
...     'width': 2702,
...     'height': 1939,
...     'count': 1,
...     'crs': CRS.from_epsg(4326),
...     'transform': Affine(0.00833333333333333, 0.0, -18.
↪175000000000068,
...                         0.0, -0.00833333333333333, 43.
↪79999999999993),
... }
```

  The meta data with the reference grid used to define the global destination grid should be first in the list, e.g., GPW population data for LitPop.
- **i_align** (*int, optional*) – Index/Position of meta in meta_list to which the global grid of the destination is to be aligned to (c.f. u_coord.align_raster_data) The default is 0.
- **target_res_arcsec** (*int, optional*) – target resolution in arcsec. The default is None, i.e. same resolution as reference data.
- **global_origins** (*tuple with two numbers (lat, lon), optional*) – global lon and lat origins as basis for destination grid. The default is the same as for GPW population data: `(-180.0, 89.99999999999991)`
- **resampling** (*resampling function, optional*) – The default is rasterio.warp.Resampling.bilinear

- **conserve** (*str, optional, either 'mean' or 'sum'*) – Conserve mean or sum of data? The default is None (no conservation).

Returns

- **data_array_list** (*list*) – contains reprojected data sets
- **meta_out** (*dict*) – contains meta data of new grid (same for all arrays)

climada.entity.exposures.litpop.litpop.**gridpoints_core_calc**(*data_arrays*, *offsets=None*, *exponents=None*, *total_val_rescale=None*)

Combines N dense numerical arrays by point-wise multipilcation and optionally rescales to new total value:

(1) An offset (1 number per array) is added to all elements in the corresponding data array in data_arrays (optional).

(2) Numbers in each array are taken to the power of the corresponding exponent (optional).

(3) Arrays are multiplied element-wise.

(4) if total_val_rescale is provided, results are normalized and re-scaled with total_val_rescale.

(5) One array with results is returned.

Parameters

- **data_arrays** (*list or array of numpy arrays containing numbers*) – Data to be combined, i.e. list containing N (min. 1) arrays of same shape.
- **total_val_rescale** (*float or int, optional*) – Total value for optional rescaling of resulting array. All values in result_array are skaled so that the sum is equal to total_val_rescale. The default (None) implies no rescaling.
- **offsets** (*list or array containing N numbers >= 0, optional*) – One numerical offset per array that is added (sum) to the corresponding array in data_arrays. The default (None) corresponds to np.zeros(N).
- **exponents** (*list or array containing N numbers >= 0, optional*) – One exponent per array used as power for the corresponding array. The default (None) corresponds to np.ones(N).

Raises

ValueError – If input lists don't have the same number of elements. Or: If arrays in data_arrays do not have the same shape.

Returns

Results from calculation described above.

Return type

np.array of same shape as arrays in data_arrays

climada.entity.exposures.litpop.nightlight module

climada.entity.exposures.litpop.nightlight.**NOAA_RESOLUTION_DEG = 0.008333333333333333**

NOAA nightlights coordinates resolution in degrees.

climada.entity.exposures.litpop.nightlight.**NASA_RESOLUTION_DEG = 0.004166666666666667**

NASA nightlights coordinates resolution in degrees.

climada.entity.exposures.litpop.nightlight.**NASA_TILE_SIZE = (21600, 21600)**

NASA nightlights tile resolution.

climada.entity.exposures.litpop.nightlight.**NOAA_BORDER = (-180, -65, 180, 75)**

NOAA nightlights border (min_lon, min_lat, max_lon, max_lat)

climada.entity.exposures.litpop.nightlight.**BM_FILENAMES = ['BlackMarble_%i_A1_geo_gray.tif', 'BlackMarble_%i_A2_geo_gray.tif', 'BlackMarble_%i_B1_geo_gray.tif', 'BlackMarble_%i_B2_geo_gray.tif', 'BlackMarble_%i_C1_geo_gray.tif', 'BlackMarble_%i_C2_geo_gray.tif', 'BlackMarble_%i_D1_geo_gray.tif', 'BlackMarble_%i_D2_geo_gray.tif']**

Nightlight NASA files which generate the whole earth when put together.

climada.entity.exposures.litpop.nightlight.**load_nasa_nl_shape**(*geometry*, *year*, *data_dir=PosixPath('/home/docs/climada/d* *dtype='float32'*)

Read nightlight data from NASA BlackMarble tiles cropped to given shape(s) and combine arrays from each tile.

1) check and download required blackmarble files
2) read and crop data from each file required in a bounding box around the given *geometry*.
3) combine data from all input files into one array. this array then contains all data in the geographic bounding box around *geometry*.
4) return array with nightlight data

Parameters

- **geometry** (*shape(s) to crop data to in degree lon/lat.*) – for example shapely.geometry.(Multi)Polygon or shapefile.Shape. from polygon defined in a shapefile. The object should have attribute 'bounds' or 'points'
- **year** (*int*) – target year for nightlight data, e.g. 2016. Closest availble year is selected.
- **data_dir** (*Path (optional)*) – Path to directory with BlackMarble data. The default is SYSTEM_DIR.
- **dtype** (*dtype*) – data type for output default 'float32', required for LitPop, choose 'int8' for integer.

Returns

- **results_array** (*numpy array*) – extracted and combined nightlight data for bounding box around shape
- **meta** (*dict*) – rasterio meta data for results_array

climada.entity.exposures.litpop.nightlight.**get_required_nl_files**(*bounds*)

Determines which of the satellite pictures are necessary for
a certain bounding box (e.g. country)

Parameters
bounds (*1x4 tuple*) – bounding box from shape (min_lon, min_lat, max_lon, max_lat).

Raises
ValueError – invalid *bounds*

Returns
req_files – Array indicating the required files for the current operation with a boolean value (1: file is required, 0: file is not required).

Return type
numpy array

climada.entity.exposures.litpop.nightlight.**check_nl_local_file_exists**(*required_files=None, check_path=PosixPath('/home/d year=2016*)

Checks if BM Satellite files are avaialbe and returns a vector denoting the missing files.

Parameters

- **required_files** (*numpy array, optional*) – boolean array of dimension (8,) with which some files can be skipped. Only files with value 1 are checked, with value zero are skipped. The default is np.ones(len(BM_FILENAMES),)
- **check_path** (*str or Path*) – absolute path where files are stored. Default: SYSTEM_DIR
- **year** (*int*) – year of the image, e.g. 2016

Returns
files_exist – Boolean array that denotes if the required files exist.

Return type
numpy array

climada.entity.exposures.litpop.nightlight.**download_nl_files**(*req_files=array([1., 1., 1., 1., 1., 1., 1., 1.]), files_exist=array([0., 0., 0., 0., 0., 0., 0., 0.]), dwnl_path=PosixPath('/home/docs/climada/d year=2016*)

Attempts to download nightlight files from NASA webpage.

Parameters

- **req_files** (*numpy array, optional*) –

  Boolean array which indicates the files required (0-> skip, 1-> download).
  The default is np.ones(len(BM_FILENAMES),).
- **files_exist** (*numpy array, optional*) –

  Boolean array which indicates if the files already
  exist locally and should not be downloaded (0-> download, 1-> skip). The default is np.zeros(len(BM_FILENAMES),).
- **dwnl_path** (*str or path, optional*) – Download directory path. The default is SYSTEM_DIR.

- **year** (*int, optional*) – Data year to be downloaded. The default is 2016.

Raises

- **ValueError** –
- **RuntimeError** –

Returns
dwnl_path – Download directory path.

Return type
str or path

`climada.entity.exposures.litpop.nightlight.`**`load_nasa_nl_shape_single_tile`**(*geometry*, *path*, *layer=0*)

Read nightlight data from single NASA BlackMarble tile and crop to given shape.

Parameters

- **geometry** (*shape or geometry object*) – shape(s) to crop data to in degree lon/lat. for example shapely.geometry.Polygon object or from polygon defined in a shapefile.
- **path** (*Path or str*) – full path to BlackMarble tif (including filename)
- **layer** (*int, optional*) – TIFF-layer to be returned. The default is 0. BlackMarble usually comes with 3 layers.

Returns

- **out_image[layer,** (*,:] : 2D numpy ndarray*) – 2d array with data cropped to bounding box of shape
- **meta** (*dict*) – rasterio meta

`climada.entity.exposures.litpop.nightlight.`**`load_nightlight_nasa`**(*bounds*, *req_files*, *year*)

Get nightlight from NASA repository that contain input boundary.

Note: Legacy for BlackMarble, not required for litpop module

Parameters

- **bounds** (*tuple*) – min_lon, min_lat, max_lon, max_lat
- **req_files** (*np.array*) – array with flags for NASA files needed
- **year** (*int*) – nightlight year

Returns

- **nightlight** (*sparse.csr_matrix*)
- **coord_nl** (*np.array*)

`climada.entity.exposures.litpop.nightlight.`**`read_bm_file`**(*bm_path*, *filename*)

Reads a single NASA BlackMarble GeoTiff and returns the data. Run all required checks first.

Note: Legacy for BlackMarble, not required for litpop module

Parameters

- **bm_path** (*str*) – absolute path where files are stored.
- **filename** (*str*) – filename of the file to be read.

Returns

- **arr1** (*array*) – Raw BM data
- **curr_file** (*gdal GeoTiff File*) – Additional info from which coordinates can be calculated.

climada.entity.exposures.litpop.nightlight.**unzip_tif_to_py**(*file_gz*)

Unzip image file, read it, flip the x axis, save values as pickle and remove tif.

Parameters
file_gz (*str*) – file fith .gz format to unzip

Returns

- **fname** (*str*) – file_name of unzipped file
- **nightlight** (*sparse.csr_matrix*)

climada.entity.exposures.litpop.nightlight.**untar_noaa_stable_nightlight**(*f_tar_ini*)

Move input tar file to SYSTEM_DIR and extract stable light file. Returns absolute path of stable light file in format tif.gz.

Parameters
f_tar_ini (*str*) – absolute path of file

Returns
f_tif_gz – path of stable light file

Return type
str

climada.entity.exposures.litpop.nightlight.**load_nightlight_noaa**(*ref_year=2013*, *sat_name=None*)

Get nightlight luminosites. Nightlight matrix, lat and lon ordered such that nightlight[1][0] corresponds to lat[1], lon[0] point (the image has been flipped).

Parameters

- **ref_year** (*int, optional*) – reference year. The default is 2013.
- **sat_name** (*str, optional*) – satellite provider (e.g. 'F10', 'F18', …)

Returns

- **nightlight** (*sparse.csr_matrix*)
- **coord_nl** (*np.array*)
- **fn_light** (*str*)

climada.entity.exposures.base module

class climada.entity.exposures.base.**Exposures**(**args*, *meta=None*, *description=None*, *ref_year=2018*, *value_unit='USD'*, *crs=None*, ***kwargs*)

Bases: `object`

geopandas GeoDataFrame with metada and columns (pd.Series) defined in Attributes.

description
metadata - description of content and origin of the data

Type
str

ref_year
metada - reference year

Type
int

value_unit
metada - unit of the exposures values

Type
str

latitude
latitude

Type
pd.Series

longitude
longitude

Type
pd.Series

value
a value for each exposure

Type
pd.Series

impf_SUFFIX
e.g. impf_TC. impact functions id for hazard TC. There might be different hazards defined: impf_TC, impf_FL, … If not provided, set to default `impf_` with ids 1 in check().

Type
pd.Series, optional

geometry
geometry of type Point of each instance. Computed in method set_geometry_points().

Type
pd.Series, optional

meta
dictionary containing corresponding raster properties (if any): width, height, crs and transform must be present at least (transform needs to contain upper left corner!). Exposures might not contain all the points of the corresponding raster. Not used in internal computations.

Type
dict

deductible
deductible value for each exposure

Type
pd.Series, optional

cover

cover value for each exposure

Type

pd.Series, optional

category_id

category id for each exposure

Type

pd.Series, optional

region_id

region id for each exposure

Type

pd.Series, optional

centr_SUFFIX

e.g. centr_TC. centroids index for hazard TC. There might be different hazards defined: centr_TC, centr_FL, … Computed in method assign_centroids().

Type

pd.Series, optional

vars_oblig = ['value', 'latitude', 'longitude']

Name of the variables needed to compute the impact.

vars_def = ['impf_', 'if_']

Name of variables that can be computed.

vars_opt = ['centr_', 'deductible', 'cover', 'category_id', 'region_id', 'geometry']

Name of the variables that aren't need to compute the impact.

property crs

Coordinate Reference System, refers to the crs attribute of the inherent GeoDataFrame

__init__(**args*, *meta=None*, *description=None*, *ref_year=2018*, *value_unit='USD'*, *crs=None*, ***kwargs*)

Creates an Exposures object from a GeoDataFrame

Parameters

- **args** – Arguments of the GeoDataFrame constructor
- **kwargs** – Named arguments of the GeoDataFrame constructor, additionally
- **meta** (*dict, optional*) – Metadata dictionary. Default: {} (empty dictionary)
- **description** (*str, optional*) – Default: None
- **ref_year** (*int, optional*) – Reference Year. Defaults to the entry of the same name in *meta* or 2018.
- **value_unit** (*str, optional*) – Unit of the exposed value. Defaults to the entry of the same name in *meta* or 'USD'.
- **crs** (*object, anything accepted by pyproj.CRS.from_user_input*) – Coordinate reference system. Defaults to the entry of the same name in *meta*, or to the CRS of the GeoDataFrame (if provided) or to 'epsg:4326'.

check()

Check Exposures consistency.

Reports missing columns in log messages. If no `impf_*` column is present in the dataframe, a default column `impf_` is added with default impact function id 1.

set_crs(*crs=None*)

Set the Coordinate Reference System. If the epxosures GeoDataFrame has a 'geometry' column it will be updated too.

Parameters

crs (*object, optional*) – anything anything accepted by pyproj.CRS.from_user_input if the original value is None it will be set to the default CRS.

set_gdf(*gdf: GeoDataFrame*, *crs=None*)

Set the *gdf* GeoDataFrame and update the CRS

Parameters

- **gdf** (*GeoDataFrame*)
- **crs** (*object, optional,*) – anything anything accepted by pyproj.CRS.from_user_input, by default None, then *gdf.crs* applies or - if not set - the exposure's current crs

get_impf_column(*haz_type=''*)

Find the best matching column name in the exposures dataframe for a given hazard type,

Parameters

haz_type (*str or None*) – hazard type, as in the hazard's.haz_type which is the HAZ_TYPE constant of the hazard's module

Returns

a column name, the first of the following that is present in the exposures' dataframe:

- `impf_[haz_type]`
- `if_[haz_type]`
- `impf_`
- `if_`

Return type

str

Raises

ValueError – if none of the above is found in the dataframe.

assign_centroids(*hazard*, *distance='euclidean'*, *threshold=100*, *overwrite=True*)

Assign for each exposure coordinate closest hazard coordinate. The Exposures `gdf` will be altered by this method. It will have an additional (or modified) column named `centr_[hazard.HAZ_TYPE]` after the call.

Uses the utility function `u_coord.match_centroids`. See there for details and parameters.

The value -1 is used for distances larger than `threshold` in point distances. In case of raster hazards the value -1 is used for centroids outside of the raster.

Parameters

- **hazard** (*Hazard*) – Hazard to match (with raster or vector centroids).

- **distance** (*str, optional*) – Distance to use in case of vector centroids. Possible values are "euclidean", "haversine" and "approx". Default: "euclidean"
- **threshold** (*float*) – If the distance (in km) to the nearest neighbor exceeds *threshold*, the index *-1* is assigned. Set *threshold* to 0, to disable nearest neighbor matching. Default: 100 (km)
- **overwrite** (*bool*) – If True, overwrite centroids already present. If False, do not assign new centroids. Default is True.

See also:

climada.util.coordinates.match_grid_points
method to associate centroids to exposure points when centroids is a raster

climada.util.coordinates.match_coordinates
method to associate centroids to exposure points

Notes

The default order of use is:

1. if centroid raster is defined, assign exposures points to the closest raster point.
2. if no raster, assign centroids to the nearest neighbor using euclidian metric

Both cases can introduce innacuracies for coordinates in lat/lon coordinates as distances in degrees differ from distances in meters on the Earth surface, in particular for higher latitude and distances larger than 100km. If more accuracy is needed, please use 'haversine' distance metric. This however is slower for (quasi-)gridded data, and works only for non-gridded data.

set_geometry_points(*scheduler=None*)

Set geometry attribute of GeoDataFrame with Points from latitude and longitude attributes.

Parameters
scheduler (*str, optional*) – used for dask map_partitions. "threads", "synchronous" or "processes"

set_lat_lon()

Set latitude and longitude attributes from geometry attribute.

set_from_raster(**args*, ***kwargs*)

This function is deprecated, use Exposures.from_raster instead.

classmethod from_raster(*file_name*, *band=1*, *src_crs=None*, *window=None*, *geometry=None*, *dst_crs=None*, *transform=None*, *width=None*, *height=None*, *resampling=Resampling.nearest*)

Read raster data and set latitude, longitude, value and meta

Parameters

- **file_name** (*str*) – file name containing values
- **band** (*int, optional*) – bands to read (starting at 1)
- **src_crs** (*crs, optional*) – source CRS. Provide it if error without it.
- **window** (*rasterio.windows.Windows, optional*) – window where data is extracted
- **geometry** (*list of shapely.geometry, optional*) – consider pixels only within these shape
- **dst_crs** (*crs, optional*) – reproject to given crs

- **transform** (*rasterio.Affine*) – affine transformation to apply
- **wdith** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampling** (*rasterio.warp,.Resampling optional*) – resampling function used for reprojection to dst_crs

Return type
Exposures

plot_scatter(*mask=None*, *ignore_zero=False*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *axis=None*, *figsize=(9, 13)*, *adapt_fontsize=True*, *title=None*, ***kwargs*)

Plot exposures geometry's value sum scattered over Earth's map. The plot will we projected according to the current crs.

Parameters

- **mask** (*np.array, optional*) – mask to apply to eai_exp plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places, by default True.
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*tuple, optional*) – figure size for plt.subplots
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **title** (*str, optional*) – a title for the plot. If not set *self.description* is used.
- **kwargs** (*optional*) – arguments for scatter matplotlib function, e.g. cmap='Greys'

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_hexbin(*mask=None*, *ignore_zero=False*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *axis=None*, *figsize=(9, 13)*, *adapt_fontsize=True*, *title=None*, ***kwargs*)

Plot exposures geometry's value sum binned over Earth's map. An other function for the bins can be set through the key reduce_C_function. The plot will we projected according to the current crs.

Parameters

- **mask** (*np.array, optional*) – mask to apply to eai_exp plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places, by default True.
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max'] Default is 'neither'.
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*tuple*) – figure size for plt.subplots Default is (9, 13).

- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **title** (*str, optional*) – a title for the plot. If not set *self.description* is used.
- **kwargs** (*optional*) – arguments for hexbin matplotlib function, e.g. *reduce_C_function=np.average*. Default is *reduce_C_function=np.sum*

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_raster(*res=None*, *raster_res=None*, *save_tiff=None*, *raster_f=<function Exposures.<lambda>>*, *label='value (log10)'*, *scheduler=None*, *axis=None*, *figsize=(9, 13)*, *fill=True*, *adapt_fontsize=True*, ***kwargs*)

Generate raster from points geometry and plot it using log10 scale *np.log10((np.fmax(raster+1, 1)))*.

Parameters

- **res** (*float, optional*) – resolution of current data in units of latitude and longitude, approximated if not provided.
- **raster_res** (*float, optional*) – desired resolution of the raster
- **save_tiff** (*str, optional*) – file name to save the raster in tiff format, if provided
- **raster_f** (*lambda function*) – transformation to use to data. Default: log10 adding 1.
- **label** (*str*) – colorbar label
- **scheduler** (*str*) – used for dask map_partitions. “threads”, “synchronous” or “processes”
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*tuple, optional*) – figure size for plt.subplots
- **fill** (*bool, optional*) – If false, the areas with no data will be plotted in white. If True, the areas with missing values are filled as 0s. The default is True.
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **kwargs** (*optional*) – arguments for imshow matplotlib function

Return type
matplotlib.figure.Figure, cartopy.mpl.geoaxes.GeoAxesSubplot

plot_basemap(*mask=None*, *ignore_zero=False*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *zoom=10*, *url={'attribution': '(C) OpenStreetMap contributors (C) CARTO', 'html_attribution': '© OpenStreetMap contributors © CARTO', 'max_zoom': 20, 'name': 'CartoDB.Positron', 'subdomains': 'abcd', 'url': 'https://{s}.basemaps.cartocdn.com/{variant}/{z}/{x}/{y}{r}.png', 'variant': 'light_all'}*, *axis=None*, ***kwargs*)

Scatter points over satellite image using contextily

Parameters

- **mask** (*np.array, optional*) – mask to apply to eai_exp plotted. Same size of the exposures, only the selected indexes will be plot.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places, by default True.

- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **zoom** (*int, optional*) – zoom coefficient used in the satellite image
- **url** (*Any, optional*) – image source, e.g., `ctx.providers.OpenStreetMap.Mapnik`. Default: `ctx.providers.CartoDB.Positron`
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for scatter matplotlib function, e.g. cmap='Greys'. Default: 'Wistia'

Return type
matplotlib.figure.Figure, cartopy.mpl.geoaxes.GeoAxesSubplot

write_hdf5(*file_name*)

Write data frame and metadata in hdf5 format

Parameters
file_name (*str*) – (path and) file name to write to.

read_hdf5(**args*, ***kwargs*)

This function is deprecated, use Exposures.from_hdf5 instead.

classmethod from_hdf5(*file_name*)

Read data frame and metadata in hdf5 format

Parameters

- **file_name** (*str*) – (path and) file name to read from.
- **additional_vars** (*list*) – list of additional variable names to read that are not in exposures.base._metadata

Return type
Exposures

read_mat(**args*, ***kwargs*)

This function is deprecated, use Exposures.from_mat instead.

classmethod from_mat(*file_name*, *var_names=None*)

Read MATLAB file and store variables in exposures.

Parameters

- **file_name** (*str*) – absolute path file
- **var_names** (*dict, optional*) – dictionary containing the name of the MATLAB variables. Default: DEF_VAR_MAT.

Return type
Exposures

to_crs(*crs=None*, *epsg=None*, *inplace=False*)

Wrapper of the `GeoDataFrame.to_crs()` method.

Transform geometries to a new coordinate reference system. Transform all geometries in a GeoSeries to a different coordinate reference system. The crs attribute on the current GeoSeries must be set. Either crs in string or dictionary form or an EPSG code may be specified for output. This method will transform all points in all objects. It has no notion or projecting entire geometries. All segments joining points are assumed to be lines in the current projection, not geodesics. Objects crossing the dateline (or other projection boundary) will have undesirable behavior.

Parameters

- **crs** (*dict or str*) – Output projection parameters as string or in dictionary form.
- **epsg** (*int*) – EPSG code specifying output projection.
- **inplace** (*bool, optional, default: False*) – Whether to return a new GeoDataFrame or do the transformation in place.

Return type
None if inplace is True else a transformed copy of the exposures object

plot (**args*, ***kwargs*)
Wrapper of the `GeoDataFrame.plot()` method

copy (*deep=True*)
Make a copy of this Exposures object.

Parameters
deep (bool) (*Make a deep copy, i.e. also copy data. Default True.*)

Return type
Exposures

write_raster (*file_name*, *value_name='value'*, *scheduler=None*)
Write value data into raster file with GeoTiff format

Parameters
file_name (*str*) – name output file in tif format

static concat (*exposures_list*)
Concatenates Exposures or DataFrame objectss to one Exposures object.

Parameters
exposures_list (*list of Exposures or DataFrames*) – The list must not be empty with the first item supposed to be an Exposures object.

Returns
with the metadata of the first item in the list and the dataframes concatenated.

Return type
Exposures

centroids_total_value (*hazard*)
Compute value of exposures close enough to be affected by hazard

Deprecated since version 3.3: This method will be removed in a future version. Use `affected_total_value()` instead.

This method computes the sum of the value of all exposures points for which a Hazard centroid is assigned.

Parameters
hazard (*Hazard*) – Hazard affecting Exposures

Returns
Sum of value of all exposures points for which a centroids is assigned

Return type
float

affected_total_value (*hazard: Hazard*, *threshold_affected: float = 0*, *overwrite_assigned_centroids: bool = True*)

Total value of the exposures that are affected by at least one hazard event (sum of value of all exposures points for which at least one event has intensity larger than the threshold).

Parameters

- **hazard** (*Hazard*) – Hazard affecting Exposures
- **threshold_affected** (*int or float*) – Hazard intensity threshold above which an exposures is considere affected. The default is 0.
- **overwrite_assigned_centroids** (*boolean*) – Assign centroids from the hazard to the exposures and overwrite existing ones. The default is True.

Returns
Sum of value of all exposures points for which a centroids is assigned and that have at least one event intensity above threshold.

Return type
float

See also:

Exposures.assign_centroids
method to assign centroids.

Note: The fraction attribute of the hazard is ignored. Thus, for hazards with fraction defined the affected values will be overestimated.

`climada.entity.exposures.base.`**`add_sea`**(*exposures*, *sea_res*, *scheduler=None*)

Add sea to geometry's surroundings with given resolution. region_id set to -1 and other variables to 0.

Parameters

- **exposures** (*Exposures*) – the Exposures object without sea surroundings.
- **sea_res** (*tuple (float,float)*) – (sea_coast_km, sea_res_km), where first parameter is distance from coast to fill with water and second parameter is resolution between sea points
- **scheduler** (*str, optional*) – used for dask map_partitions. "threads", "synchronous" or "processes"

Return type
Exposures

`climada.entity.exposures.base.`**`INDICATOR_IMPF = 'impf_'`**

Name of the column containing the impact functions id of specified hazard

`climada.entity.exposures.base.`**`INDICATOR_CENTR = 'centr_'`**

Name of the column containing the centroids id of specified hazard

5.2.3 climada.entity.impact_funcs package

climada.entity.impact_funcs.base module

class climada.entity.impact_funcs.base.**ImpactFunc**(*haz_type: str = '', id: str | int = '', intensity: ndarray | None = None, mdd: ndarray | None = None, paa: ndarray | None = None, intensity_unit: str = '', name: str = ''*)

Bases: object

Contains the definition of one impact function.

haz_type

hazard type acronym (e.g. 'TC')

Type
str

id

id of the impact function. Exposures of the same type will refer to the same impact function id

Type
int or str

name

name of the ImpactFunc

Type
str

intensity_unit

unit of the intensity

Type
str

intensity

intensity values

Type
np.array

mdd

mean damage (impact) degree for each intensity (numbers in [0,1])

Type
np.array

paa

percentage of affected assets (exposures) for each intensity (numbers in [0,1])

Type
np.array

__init__(*haz_type: str = '', id: str | int = '', intensity: ndarray | None = None, mdd: ndarray | None = None, paa: ndarray | None = None, intensity_unit: str = '', name: str = ''*)

Initialization.

Parameters

- **haz_type** (*str, optional*) – Hazard type acronym (e.g. 'TC').
- **id** (*int or str, optional*) – id of the impact function. Exposures of the same type will refer to the same impact function id.
- **intensity** (*np.array, optional*) – Intensity values. Defaults to empty array.
- **mdd** (*np.array, optional*) – Mean damage (impact) degree for each intensity (numbers in [0,1]). Defaults to empty array.
- **paa** (*np.array, optional*) – Percentage of affected assets (exposures) for each intensity (numbers in [0,1]). Defaults to empty array.
- **intensity_unit** (*str, optional*) – Unit of the intensity.
- **name** (*str, optional*) – Name of the ImpactFunc.

calc_mdr(*inten: float | ndarray*) → ndarray

Interpolate impact function to a given intensity.

Parameters

inten (*float or np.array*) – intensity, the x-coordinate of the interpolated values.

Return type

np.array

plot(*axis=None*, ***kwargs*)

Plot the impact functions MDD, MDR and PAA in one graph, where MDR = PAA * MDD.

Parameters

- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for plot matplotlib function, e.g. marker='x'

Return type

matplotlib.axes._subplots.AxesSubplot

check()

Check consistent instance data.

Raises

`ValueError` –

classmethod from_step_impf(*intensity: tuple[float, float, float]*, *haz_type: str*, *mdd: tuple[float, float] = (0, 1)*, *paa: tuple[float, float] = (1, 1)*, *impf_id: int = 1*, ***kwargs*)

Step function type impact function.

By default, everything is destroyed above the step. Useful for high resolution modelling.

This method modifies self (climada.entity.impact_funcs instance) by assigning an id, intensity, mdd and paa to the impact function.

Parameters

- **intensity** (*tuple(float, float, float)*) – tuple of 3-intensity numbers: (minimum, threshold, maximum)
- **haz_type** (*str*) – the reference string for the hazard (e.g., 'TC', 'RF', 'WS', …)
- **mdd** (*tuple(float, float)*) – (min, max) mdd values. The default is (0, 1)
- **paa** (*tuple(float, float)*) – (min, max) paa values. The default is (1, 1)
- **impf_id** (*int, optional, default=1*) – impact function id

- **kwargs** – keyword arguments passed to ImpactFunc()

Returns
impf – Step impact function

Return type
climada.entity.impact_funcs.ImpactFunc

set_step_impf(**args*, ***kwargs*)

This function is deprecated, use ImpactFunc.from_step_impf instead.

classmethod from_sigmoid_impf(*intensity: tuple[float, float, float]*, *L: float*, *k: float*, *x0: float*, *haz_type: str*, *impf_id: int = 1*, ***kwargs*)

Sigmoid type impact function hinging on three parameter.

This type of impact function is very flexible for any sort of study, hazard and resolution. The sigmoid is defined as:

$$f(x) = \frac{L}{1 + exp^{-k(x-x0)}}$$

For more information: https://en.wikipedia.org/wiki/Logistic_function

This method modifies self (climada.entity.impact_funcs instance) by assining an id, intensity, mdd and paa to the impact function.

Parameters

- **intensity** (*tuple(float, float, float)*) – tuple of 3 intensity numbers along np.arange(min, max, step)
- **L** (*float*) – "top" of sigmoid
- **k** (*float*) – "slope" of sigmoid
- **x0** (*float*) – intensity value where f(x)==L/2
- **haz_type** (*str*) – the reference string for the hazard (e.g., 'TC', 'RF', 'WS', …)
- **impf_id** (*int, optional, default=1*) – impact function id
- **kwargs** – keyword arguments passed to ImpactFunc()

Returns
impf – Sigmoid impact function

Return type
climada.entity.impact_funcs.ImpactFunc

set_sigmoid_impf(**args*, ***kwargs*)

This function is deprecated, use LitPop.from_countries instead.

climada.entity.impact_funcs.impact_func_set module

class climada.entity.impact_funcs.impact_func_set.**ImpactFuncSet**(*impact_funcs: Iterable[ImpactFunc] | None = None*)

Bases: object

Contains impact functions of type ImpactFunc. Loads from files with format defined in FILE_EXT.

_data

contains ImpactFunc classes. It's not suppossed to be directly accessed. Use the class methods instead.

Type

dict

__init__(*impact_funcs: Iterable[ImpactFunc] | None = None*)

Initialization.

Build an impact function set from an iterable of ImpactFunc.

Parameters

impact_funcs (*iterable of ImpactFunc, optional*) – An iterable (list, set, array, …) of ImpactFunc.

Examples

Fill impact functions with values and check consistency data:

```
>>> intensity = np.array([0, 20])
>>> paa = np.array([0, 1])
>>> mdd = np.array([0, 0.5])
>>> fun_1 = ImpactFunc("TC", 3, intensity, mdd, paa)
>>> imp_fun = ImpactFuncSet([fun_1])
>>> imp_fun.check()
```

Read impact functions from file and check data consistency.

```
>>> imp_fun = ImpactFuncSet.from_excel(ENT_TEMPLATE_XLS)
```

clear()

Reinitialize attributes.

append(*func*)

Append a ImpactFunc. Overwrite existing if same id and haz_type.

Parameters

func (*ImpactFunc*) – ImpactFunc instance

Raises

ValueError –

remove_func(*haz_type=None, fun_id=None*)

Remove impact function(s) with provided hazard type and/or id. If no input provided, all impact functions are removed.

Parameters

- **haz_type** (*str, optional*) – all impact functions with this hazard
- **fun_id** (*int, optional*) – all impact functions with this id

get_func(*haz_type=None, fun_id=None*)

Get ImpactFunc(s) of input hazard type and/or id. If no input provided, all impact functions are returned.

Parameters

- **haz_type** (*str, optional*) – hazard type
- **fun_id** (*int, optional*) – ImpactFunc id

Returns

- *ImpactFunc (if haz_type and fun_id),*
- *list(ImpactFunc) (if haz_type or fun_id),*
- **{ImpactFunc.haz_type** (*{ImpactFunc.id : ImpactFunc}} (if None)*)

get_hazard_types(*fun_id=None*)

Get impact functions hazard types contained for the id provided. Return all hazard types if no input id.

Parameters
fun_id (*int, optional*) – id of an impact function

Return type
list(str)

get_ids(*haz_type=None*)

Get impact functions ids contained for the hazard type provided. Return all ids for each hazard type if no input hazard type.

Parameters
haz_type (*str, optional*) – hazard type from which to obtain the ids

Returns

- *list(ImpactFunc.id) (if haz_type provided),*
- **{ImpactFunc.haz_type** (*list(ImpactFunc.id)} (if no haz_type)*)

size(*haz_type=None*, *fun_id=None*)

Get number of impact functions contained with input hazard type and /or id. If no input provided, get total number of impact functions.

Parameters

- **haz_type** (*str, optional*) – hazard type
- **fun_id** (*int, optional*) – ImpactFunc id

Return type
int

check()

Check instance attributes.

Raises
ValueError –

extend(*impact_funcs*)

Append impact functions of input ImpactFuncSet to current ImpactFuncSet. Overwrite ImpactFunc if same id and haz_type.

Parameters
impact_funcs (*ImpactFuncSet*) – ImpactFuncSet instance to extend

Raises
ValueError –

plot(*haz_type=None*, *fun_id=None*, *axis=None*, ***kwargs*)

Plot impact functions of selected hazard (all if not provided) and selected function id (all if not provided).

Parameters

- **haz_type** (*str, optional*) – hazard type

- **fun_id** (*int, optional*) – id of the function

Return type
matplotlib.axes._subplots.AxesSubplot

classmethod from_excel(*file_name*, *var_names=None*)
Read excel file following template and store variables.

Parameters

- **file_name** (*str*) – absolute file name
- **description** (*str, optional*) – description of the data
- **var_names** (*dict, optional*) – name of the variables in the file

Return type
ImpactFuncSet

read_excel(**args*, ***kwargs*)
This function is deprecated, use ImpactFuncSet.from_excel instead.

classmethod from_mat(*file_name*, *var_names=None*)
Read MATLAB file generated with previous MATLAB CLIMADA version.

Parameters

- **file_name** (*str*) – absolute file name
- **description** (*str, optional*) – description of the data
- **var_names** (*dict, optional*) – name of the variables in the file

Returns
impf_set – Impact func set as defined in matlab file.

Return type
climada.entity.impact_func_set.ImpactFuncSet

read_mat(**args*, ***kwargs*)
This function is deprecated, use ImpactFuncSet.from_mat instead.

write_excel(*file_name*, *var_names=None*)
Write excel file following template.

Parameters

- **file_name** (*str*) – absolute file name to write
- **var_names** (*dict, optional*) – name of the variables in the file

climada.entity.impact_funcs.storm_europe module

class climada.entity.impact_funcs.storm_europe.**ImpfStormEurope**
Bases: *ImpactFunc*

Impact functions for tropical cyclones.

__init__()
Initialization.

Parameters

- **haz_type** (*str, optional*) – Hazard type acronym (e.g. 'TC').

- **id** (*int or str, optional*) – id of the impact function. Exposures of the same type will refer to the same impact function id.
- **intensity** (*np.array, optional*) – Intensity values. Defaults to empty array.
- **mdd** (*np.array, optional*) – Mean damage (impact) degree for each intensity (numbers in [0,1]). Defaults to empty array.
- **paa** (*np.array, optional*) – Percentage of affected assets (exposures) for each intensity (numbers in [0,1]). Defaults to empty array.
- **intensity_unit** (*str, optional*) – Unit of the intensity.
- **name** (*str, optional*) – Name of the ImpactFunc.

`classmethod from_schwierz(impf_id=1)`

Generate the impact function of Schwierz et al. 2010, doi:10.1007/s10584-009-9712-1

Returns
impf – impact function for asset damages due to storm defined in Schwierz et al. 2010

Return type
climada.entity.impact_funcs.storm_europe.ImpfStormEurope:

`classmethod from_welker(impf_id=1)`

Return the impact function of Welker et al. 2021, doi:10.5194/nhess-21-279-2021 It is the Schwierz function, calibrated with a simple multiplicative factor to minimize RMSE between modelled damages and reported damages.

Returns
impf – impact function for asset damages due to storm defined in Welker et al. 2021

Return type
climada.entity.impact_funcs.storm_europe.ImpfStormEurope:

`set_schwierz(impf_id=1)`

This function is deprecated, use ImpfStormEurope.from_schwierz instead.

`set_welker(impf_id=1)`

This function is deprecated, use ImpfStormEurope.from_welker instead.

`climada.entity.impact_funcs.storm_europe.IFStormEurope()`

Is ImpfStormEurope now

Deprecated since version The: class name IFStormEurope is deprecated and won't be supported in a future version. Use ImpfStormEurope instead

climada.entity.impact_funcs.trop_cyclone module

`class climada.entity.impact_funcs.trop_cyclone.ImpfTropCyclone`

Bases: *ImpactFunc*

Impact functions for tropical cyclones.

`__init__()`

Initialization.

Parameters

- **haz_type** (*str, optional*) – Hazard type acronym (e.g. 'TC').

- **id** (*int or str, optional*) – id of the impact function. Exposures of the same type will refer to the same impact function id.
- **intensity** (*np.array, optional*) – Intensity values. Defaults to empty array.
- **mdd** (*np.array, optional*) – Mean damage (impact) degree for each intensity (numbers in [0,1]). Defaults to empty array.
- **paa** (*np.array, optional*) – Percentage of affected assets (exposures) for each intensity (numbers in [0,1]). Defaults to empty array.
- **intensity_unit** (*str, optional*) – Unit of the intensity.
- **name** (*str, optional*) – Name of the ImpactFunc.

set_emanuel_usa(**args*, ***kwargs*)

This function is deprecated, use from_emanuel_usa() instead.

classmethod from_emanuel_usa(*impf_id=1*, *intensity=array([0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120])*, *v_thresh=25.7*, *v_half=74.7*, *scale=1.0*)

Init TC impact function using the formula of Kerry Emanuel, 2011: 'Global Warming Effects on U.S. Hurricane Damage', https://doi.org/10.1175/WCAS-D-11-00007.1

Parameters

- **impf_id** (*int, optional*) – impact function id. Default: 1
- **intensity** (*np.array, optional*) – intensity array in m/s. Default: 5 m/s step array from 0 to 120m/s
- **v_thresh** (*float, optional*) – first shape parameter, wind speed in m/s below which there is no damage. Default: 25.7(Emanuel 2011)
- **v_half** (*float, optional*) – second shape parameter, wind speed in m/s at which 50% of max. damage is expected. Default: v_threshold + 49 m/s (mean value of Sealy & Strobl 2017)
- **scale** (*float, optional*) – scale parameter, linear scaling of MDD. 0<=scale<=1. Default: 1.0

Raises

`ValueError` –

Returns

impf – TC impact function instance based on formula by Emanuel (2011)

Return type

ImpfTropCyclone

class climada.entity.impact_funcs.trop_cyclone.**ImpfSetTropCyclone**

Bases: *ImpactFuncSet*

Impact function set (ImpfS) for tropical cyclones.

__init__()

Initialization.

Build an impact function set from an iterable of ImpactFunc.

Parameters

impact_funcs (*iterable of ImpactFunc, optional*) – An iterable (list, set, array, …) of ImpactFunc.

Examples

Fill impact functions with values and check consistency data:

```
>>> intensity = np.array([0, 20])
>>> paa = np.array([0, 1])
>>> mdd = np.array([0, 0.5])
>>> fun_1 = ImpactFunc("TC", 3, intensity, mdd, paa)
>>> imp_fun = ImpactFuncSet([fun_1])
>>> imp_fun.check()
```

Read impact functions from file and check data consistency.

```
>>> imp_fun = ImpactFuncSet.from_excel(ENT_TEMPLATE_XLS)
```

set_calibrated_regional_ImpfSet (**args*, ***kwargs*)

This function is deprecated, use from_calibrated_regional_ImpfSet() instead.

classmethod from_calibrated_regional_ImpfSet (*calibration_approach='TDR'*, *q=0.5*, *input_file_path=None*, *version=1*)

Calibrated regional TC wind impact functions

Based on Eberenz et al. 2021: https://doi.org/10.5194/nhess-21-393-2021

Parameters

- **calibration_approach** (*str, optional*) – The following values are supported:

  'TDR' (default)
  Total damage ratio (TDR) optimization with TDR=1.0 (simulated damage = reported damage from EM-DAT)

  'TDR1.5'
  Total damage ratio (TDR) optimization with TDR=1.5 (simulated damage = 1.5*reported damage from EM-DAT)

  'RMSF'
  Root-mean-squared fraction (RMSF) optimization

  'EDR'
  quantile from individually fitted v_half per event, i.e. v_half fitted to get EDR=1.0 for each event

- **q** (*float, optional*) – Quantile between 0 and 1.0 to select (EDR only). Default: 0.5, i.e. median v_half
- **input_file_path** (*str or DataFrame, optional*) – full path to calibration result file to be used instead of default file in repository (expert users only)

Returns

impf_set – TC Impact Function Set based on Eberenz et al, 2021.

Return type

ImpfSetTropCyclone

static calibrated_regional_vhalf (*calibration_approach='TDR'*, *q=0.5*, *input_file_path=None*, *version=1*)

Calibrated TC wind impact function slope parameter v_half per region

Based on Eberenz et al., 2021: https://doi.org/10.5194/nhess-21-393-2021

Parameters

- **calibration_approach** (*str, optional*) – The following values are supported:

  'TDR' (default)
  Total damage ratio (TDR) optimization with TDR=1.0 (simulated damage = reported damage from EM-DAT)

  'TDR1.5'
  Total damage ratio (TDR) optimization with TDR=1.5 (simulated damage = 1.5*reported damage from EM-DAT)

  'RMSF'
  Root-mean-squared fraction (RMSF) optimization

  'EDR'
  quantile from individually fitted v_half per event, i.e. v_half fitted to get EDR=1.0 for each event

- **q** (*float, optional*) – Quantile between 0 and 1.0 to select (EDR only). Default: 0.5, i.e. median v_half
- **input_file_path** (*str or DataFrame, optional*) – full path to calibration result file to be used instead of default file in repository (expert users only)

Raises
`ValueError` –

Returns
v_half – TC impact function slope parameter v_half per region

Return type
dict

`static get_countries_per_region`(*region=None*)

Returns dictionaries with numerical and alphabetical ISO3 codes of all countries associated to a calibration region. Only contains countries that were affected by tropical cyclones between 1980 and 2017 according to EM-DAT.

Parameters
region (*str*) – regional abbreviation (default='all'), either 'NA1', 'NA2', 'NI', 'OC', 'SI', 'WP1', 'WP2', 'WP3', 'WP4', or 'all'.

Returns

- **region_name** (*dict or str*) – long name per region
- **impf_id** (*dict or int*) – impact function ID per region
- **iso3n** (*dict or list*) – numerical ISO3codes (=region_id) per region
- **iso3a** (*dict or list*) – numerical ISO3codes (=region_id) per region

`climada.entity.impact_funcs.trop_cyclone.IFTropCyclone()`

Is ImpfTropCyclone now

Deprecated since version The: class name IFTropCyclone is deprecated and won't be supported in a future version. Use ImpfTropCyclone instead

5.2.4 climada.entity.measures package

climada.entity.measures.base module

class climada.entity.measures.base.**Measure**(*name: str = '', haz_type: str = '', cost: float = 0, hazard_set: str = 'nil', hazard_freq_cutoff: float = 0, exposures_set: str = 'nil', imp_fun_map: str = 'nil', hazard_inten_imp: Tuple[float, float] = (1, 0), mdd_impact: Tuple[float, float] = (1, 0), paa_impact: Tuple[float, float] = (1, 0), exp_region_id: list | None = None, risk_transf_attach: float = 0, risk_transf_cover: float = 0, risk_transf_cost_factor: float = 1, color_rgb: ndarray | None = None*)

Bases: object

Contains the definition of one measure.

name

name of the measure

Type

str

haz_type

related hazard type (peril), e.g. TC

Type

str

color_rgb

integer array of size 3. Color code of this measure in RGB

Type

np.array

cost

discounted cost (in same units as assets)

Type

float

hazard_set

file name of hazard to use (in h5 format)

Type

str

hazard_freq_cutoff

hazard frequency cutoff

Type

float

exposures_set

file name of exposure to use (in h5 format) or Exposure instance

Type

str or climada.entity.Exposure

imp_fun_map

change of impact function id of exposures, e.g. '1to3'

Type
str

hazard_inten_imp

parameter a and b of hazard intensity change

Type
tuple(float, float)

mdd_impact

parameter a and b of the impact over the mean damage degree

Type
tuple(float, float)

paa_impact

parameter a and b of the impact over the percentage of affected assets

Type
tuple(float, float)

exp_region_id

region id of the selected exposures to consider ALL the previous parameters

Type
int

risk_transf_attach

risk transfer attachment

Type
float

risk_transf_cover

risk transfer cover

Type
float

risk_transf_cost_factor

factor to multiply to resulting insurance layer to get the total cost of risk transfer

Type
float

__init__(*name: str = ''*, *haz_type: str = ''*, *cost: float = 0*, *hazard_set: str = 'nil'*, *hazard_freq_cutoff: float = 0*, *exposures_set: str = 'nil'*, *imp_fun_map: str = 'nil'*, *hazard_inten_imp: Tuple[float, float] = (1, 0)*, *mdd_impact: Tuple[float, float] = (1, 0)*, *paa_impact: Tuple[float, float] = (1, 0)*, *exp_region_id: list | None = None*, *risk_transf_attach: float = 0*, *risk_transf_cover: float = 0*, *risk_transf_cost_factor: float = 1*, *color_rgb: ndarray | None = None*)

Initialize a Measure object with given values.

Parameters

- **name** (*str, optional*) – name of the measure
- **haz_type** (*str, optional*) – related hazard type (peril), e.g. TC
- **cost** (*float, optional*) – discounted cost (in same units as assets)

- **hazard_set** (*str, optional*) – file name of hazard to use (in h5 format)
- **hazard_freq_cutoff** (*float, optional*) – hazard frequency cutoff
- **exposures_set** (*str or climada.entity.Exposure, optional*) – file name of exposure to use (in h5 format) or Exposure instance
- **imp_fun_map** (*str, optional*) – change of impact function id of exposures, e.g. '1to3'
- **hazard_inten_imp** (*tuple(float, float), optional*) – parameter a and b of hazard intensity change
- **mdd_impact** (*tuple(float, float), optional*) – parameter a and b of the impact over the mean damage degree
- **paa_impact** (*tuple(float, float), optional*) – parameter a and b of the impact over the percentage of affected assets
- **exp_region_id** (*int, optional*) – region id of the selected exposures to consider ALL the previous parameters
- **risk_transf_attach** (*float, optional*) – risk transfer attachment
- **risk_transf_cover** (*float, optional*) – risk transfer cover
- **risk_transf_cost_factor** (*float, optional*) – factor to multiply to resulting insurance layer to get the total cost of risk transfer
- **color_rgb** (*np.array, optional*) – integer array of size 3. Color code of this measure in RGB. Default is None (corresponds to black).

`check``()`

Check consistent instance data.

Raises

`ValueError` –

`calc_impact``(`*exposures*, *imp_fun_set*, *hazard*, *assign_centroids=True*`)`

Apply measure and compute impact and risk transfer of measure implemented over inputs.

Parameters

- **exposures** (*climada.entity.Exposures*) – exposures instance
- **imp_fun_set** (*climada.entity.ImpactFuncSet*) – impact function set instance
- **hazard** (*climada.hazard.Hazard*) – hazard instance
- **assign_centroids** (*bool, optional*) – indicates whether centroids are assigned to the self.exposures object. Centroids assignment is an expensive operation; set this to `False` to save computation time if the hazards' centroids are already assigned to the exposures object. Default: True

Returns

resulting impact and risk transfer of measure

Return type

climada.engine.Impact

`apply``(`*exposures*, *imp_fun_set*, *hazard*`)`

Implement measure with all its defined parameters.

Parameters

- **exposures** (*climada.entity.Exposures*) – exposures instance

- **imp_fun_set** (*climada.entity.ImpactFuncSet*) – impact function set instance
- **hazard** (*climada.hazard.Hazard*) – hazard instance

Returns

- **new_exp** (*climada.entity.Exposure*) – Exposure with implemented measure with all defined parameters
- **new_ifs** (*climada.entity.ImpactFuncSet*) – Impact function set with implemented measure with all defined parameters
- **new_haz** (*climada.hazard.Hazard*) – Hazard with implemented measure with all defined parameters

climada.entity.measures.measure_set module

class climada.entity.measures.measure_set.**MeasureSet**(*measure_list: List[Measure] | None = None*)

Bases: object

Contains measures of type Measure. Loads from files with format defined in FILE_EXT.

_data

Contains Measure objects. This attribute is not suppossed to be accessed directly. Use the available methods instead.

Type
dict

__init__(*measure_list: List[Measure] | None = None*)

Initialize a new MeasureSet object with specified data.

Parameters
measure_list (*list of Measure objects, optional*) – The measures to include in the MeasureSet

Examples

Fill MeasureSet with values and check consistency data:

```
>>> act_1 = Measure(
...     name='Seawall',
...     color_rgb=np.array([0.1529, 0.2510, 0.5451]),
...     hazard_intensity=(1, 0),
...     mdd_impact=(1, 0),
...     paa_impact=(1, 0),
... )
>>> meas = MeasureSet([act_1])
>>> meas.check()
```

Read measures from file and checks consistency data:

```
>>> meas = MeasureSet.from_excel(ENT_TEMPLATE_XLS)
```

clear(*_data: dict | None = None*)

Reinitialize attributes.

Parameters
_data (*dict, optional*) – A dict containing the Measure objects. For internal use only: It's not suppossed to be set directly. Use the class methods instead.

append(*meas*)
Append an Measure. Override if same name and haz_type.

Parameters
meas (*Measure*) – Measure instance

Raises
ValueError –

remove_measure(*haz_type=None*, *name=None*)
Remove impact function(s) with provided hazard type and/or id. If no input provided, all impact functions are removed.

Parameters

- **haz_type** (*str, optional*) – all impact functions with this hazard
- **name** (*str, optional*) – measure name

get_measure(*haz_type=None*, *name=None*)
Get ImpactFunc(s) of input hazard type and/or id. If no input provided, all impact functions are returned.

Parameters

- **haz_type** (*str, optional*) – hazard type
- **name** (*str, optional*) – measure name

Returns

- *Measure (if haz_type and name),*
- *list(Measure) (if haz_type or name),*
- **{Measure.haz_type** (*{Measure.name : Measure}} (if None)*)

get_hazard_types(*meas=None*)
Get measures hazard types contained for the name provided. Return all hazard types if no input name.

Parameters
name (*str, optional*) – measure name

Return type
list(str)

get_names(*haz_type=None*)
Get measures names contained for the hazard type provided. Return all names for each hazard type if no input hazard type.

Parameters
haz_type (*str, optional*) – hazard type from which to obtain the names

Returns

- *list(Measure.name) (if haz_type provided),*
- **{Measure.haz_type** (*list(Measure.name)} (if no haz_type)*)

size(*haz_type=None*, *name=None*)

Get number of measures contained with input hazard type and /or id. If no input provided, get total number of impact functions.

Parameters

- **haz_type** (*str, optional*) – hazard type
- **name** (*str, optional*) – measure name

Return type

int

check()

Check instance attributes.

Raises

ValueError –

extend(*meas_set*)

Extend measures of input MeasureSet to current MeasureSet. Overwrite Measure if same name and haz_type.

Parameters

impact_funcs (*MeasureSet*) – ImpactFuncSet instance to extend

Raises

ValueError –

classmethod from_mat(*file_name*, *var_names=None*)

Read MATLAB file generated with previous MATLAB CLIMADA version.

Parameters

- **file_name** (*str*) – absolute file name
- **description** (*str, optional*) – description of the data
- **var_names** (*dict, optional*) – name of the variables in the file

Returns

meas_set – Measure Set from matlab file

Return type

climada.entity.MeasureSet()

read_mat(**args*, ***kwargs*)

This function is deprecated, use MeasureSet.from_mat instead.

classmethod from_excel(*file_name*, *var_names=None*)

Read excel file following template and store variables.

Parameters

- **file_name** (*str*) – absolute file name
- **description** (*str, optional*) – description of the data
- **var_names** (*dict, optional*) – name of the variables in the file

Returns

meas_set – Measures set from Excel

Return type

climada.entity.MeasureSet

read_excel(**args*, ***kwargs*)

This function is deprecated, use MeasureSet.from_excel instead.

write_excel(*file_name*, *var_names=None*)

Write excel file following template.

Parameters

- **file_name** (*str*) – absolute file name to write
- **var_names** (*dict, optional*) – name of the variables in the file

5.2.5 climada.entity.entity_def module

class climada.entity.entity_def.**Entity**(*exposures:* Exposures | *None = None*, *disc_rates:* DiscRates | *None = None*, *impact_func_set:* ImpactFuncSet | *None = None*, *measure_set:* MeasureSet | *None = None*)

Bases: object

Collects exposures, impact functions, measures and discount rates. Default values set when empty constructor.

exposures

exposures

Type

Exposures

impact_funcs

impact functions set

Type

ImpactFuncSet

measures

measures

Type

MeasureSet

disc_rates

discount rates

Type

DiscRates

def_file

Default file from configuration file

Type

str

__init__(*exposures:* Exposures | *None = None*, *disc_rates:* DiscRates | *None = None*, *impact_func_set:* ImpactFuncSet | *None = None*, *measure_set:* MeasureSet | *None = None*)

Initialize entity

Parameters

- **exposures** (*climada.entity.Exposures, optional*) – Exposures of the entity. The default is None (empty Exposures()).

- **disc_rates** (*climada.entity.DiscRates, optional*) – Disc rates of the entity. The default is None (empty DiscRates()).
- **impact_func_set** (*climada.entity.ImpactFuncSet, optional*) – The impact function set. The default is None (empty ImpactFuncSet()).
- **measure_set** (*climada.entity.Measures, optional*) – The measures. The default is None (empty MeasuresSet().

classmethod from_mat(*file_name*)

Read MATLAB file of climada.

Parameters
file_name (*str, optional*) – file name(s) or folder name containing the files to read

Returns
ent – The entity from matlab file

Return type
climada.entity.Entity

read_mat(**args*, ***kwargs*)

This function is deprecated, use Entity.from_mat instead.

classmethod from_excel(*file_name*)

Read csv or xls or xlsx file following climada's template.

Parameters

- **file_name** (*str, optional*) – file name(s) or folder name containing the files to read
- **description** (*str or list(str), optional*) – one description of the data or a description of each data file

Returns
ent – The entity from excel file

Return type
climada.entity.Entity

read_excel(**args*, ***kwargs*)

This function is deprecated, use Entity.from_excel instead.

write_excel(*file_name*)

Write excel file following template.

check()

Check instance attributes.

Raises
ValueError –

5.2.6 climada.entity.tag module

class climada.entity.tag.**Tag**(**args*, ***kwargs*)

Bases: `Tag`

kept for backwards compatibility with climada <= 3.3

__init__(**args*, ***kwargs*)

Deprecated since version This: class is not supported anymore and will be removed in the next version of climada.

5.3 climada.hazard package

5.3.1 climada.hazard.centroids package

climada.hazard.centroids.centr module

class climada.hazard.centroids.centr.**Centroids**(*lat: ndarray | None = None*, *lon: ndarray | None = None*, *geometry: GeoSeries | None = None*, *meta: Dict[Any, Any] | None = None*, *area_pixel: ndarray | None = None*, *on_land: ndarray | None = None*, *region_id: ndarray | None = None*, *elevation: ndarray | None = None*, *dist_coast: ndarray | None = None*)

Bases: `object`

Contains raster or vector centroids.

meta

rasterio meta dictionary containing raster properties: width, height, crs and transform must be present at least. The affine ransformation needs to be shearless (only stretching) and have positive x- and negative y-orientation.

Type
dict, optional

lat

latitude of size size

Type
np.array, optional

lon

longitude of size size

Type
np.array, optional

geometry

contains lat and lon crs. Might contain geometry points for lat and lon

Type
gpd.GeoSeries, optional

area_pixel

area of size size

Type

np.array, optional

dist_coast

distance to coast of size size

Type

np.array, optional

on_land

on land (True) and on sea (False) of size size

Type

np.array, optional

region_id

country region code of size size

Type

np.array, optional

elevation

elevation of size size

Type

np.array, optional

vars_check = {'area_pixel', 'dist_coast', 'elevation', 'geometry', 'lat', 'lon', 'on_land', 'region_id'}

Variables whose size will be checked

__init__(*lat: ndarray | None = None*, *lon: ndarray | None = None*, *geometry: GeoSeries | None = None*, *meta: Dict[Any, Any] | None = None*, *area_pixel: ndarray | None = None*, *on_land: ndarray | None = None*, *region_id: ndarray | None = None*, *elevation: ndarray | None = None*, *dist_coast: ndarray | None = None*)

Initialization

Parameters

- **lat** (*np.array, optional*) – latitude of size size. Defaults to empty array
- **lon** (*np.array, optional*) – longitude of size size. Defaults to empty array
- **geometry** (*gpd.GeoSeries, optional*) – contains lat and lon crs. Might contain geometry points for lat and lon. Defaults to empty gpd.Geoseries with crs=DEF_CRS
- **meta** (*dict, optional*) – rasterio meta dictionary containing raster properties: width, height, crs and transform must be present at least. The affine ransformation needs to be shearless (only stretching) and have positive x- and negative y-orientation. Defaults to empty dict()
- **area_pixel** (*np.array, optional*) – area of size size. Defaults to empty array
- **on_land** (*np.array, optional*) – on land (True) and on sea (False) of size size. Defaults to empty array
- **region_id** (*np.array, optional*) – country region code of size size, Defaults to empty array
- **elevation** (*np.array, optional*) – elevation of size size. Defaults to empty array
- **dist_coast** (*np.array, optional*) – distance to coast of size size. Defaults to empty array

check()

Check integrity of stored information.

Checks that either *meta* attribute is set, or *lat*, *lon* and *geometry.crs*. Checks sizes of (optional) data attributes.

equal(*centr*)

Return True if two centroids equal, False otherwise

Parameters
centr (*Centroids*) – centroids to compare

Returns
eq

Return type
bool

static from_base_grid(*land=False*, *res_as=360*, *base_file=None*)

Initialize from base grid data provided with CLIMADA

Parameters

- **land** (*bool, optional*) – If True, restrict to grid points on land. Default: False.
- **res_as** (*int, optional*) – Base grid resolution in arc-seconds (one of 150, 360). Default: 360.
- **base_file** (*str, optional*) – If set, read this file instead of one provided with climada.

classmethod from_geodataframe(*gdf*, *geometry_alias='geom'*)

Create Centroids instance from GeoDataFrame.

Deprecated since version 3.3: This method will be removed in a future version. Pass the data you want to construct the Centroids with to the constructor instead.

The geometry, lat, and lon attributes are set from the GeoDataFrame.geometry attribute, while the columns are copied as attributes to the Centroids object in the form of numpy.ndarrays using pandas.Series.to_numpy. The Series dtype will thus be respected.

Columns named lat or lon are ignored, as they would overwrite the coordinates extracted from the point features. If the geometry attribute bears an alias, it can be dropped by setting the geometry_alias parameter.

If the GDF includes a region_id column, but no on_land column, then on_land=True is inferred for those centroids that have a set region_id.

Example

```
>>> gdf = geopandas.read_file('centroids.shp')
>>> gdf.region_id = gdf.region_id.astype(int)  # type coercion
>>> centroids = Centroids.from_geodataframe(gdf)
```

Parameters

- **gdf** (*GeoDataFrame*) – Where the geometry column needs to consist of point features. See above for details on processing.
- **geometry_alias** (*str, opt*) – Alternate name for the geometry column; dropped to avoid duplicate assignment.

Returns
centr – Centroids with data from given GeoDataFrame

Return type
Centroids

classmethod from_pix_bounds(*xf_lat*, *xo_lon*, *d_lat*, *d_lon*, *n_lat*, *n_lon*, *crs='EPSG:4326'*)

Create Centroids object with meta attribute according to pixel border data.

Deprecated since version 3.3: This method will be removed in a future version. CLIMADA will only support regular grids with a constant lat/lon resolution then. Use `from_pnt_bounds()` instead.

Parameters

- **xf_lat** (*float*) – upper latitude (top)
- **xo_lon** (*float*) – left longitude
- **d_lat** (*float*) – latitude step (negative)
- **d_lon** (*float*) – longitude step (positive)
- **n_lat** (*int*) – number of latitude points
- **n_lon** (*int*) – number of longitude points
- **crs** (*dict() or rasterio.crs.CRS, optional*) – CRS. Default: DEF_CRS

Returns
centr – Centroids with meta according to given pixel border data.

Return type
Centroids

set_raster_from_pnt_bounds(**args*, ***kwargs*)

This function is deprecated, use Centroids.from_pnt_bounds instead.

classmethod from_pnt_bounds(*points_bounds*, *res*, *crs='EPSG:4326'*)

Create Centroids object with meta attribute according to points border data.

raster border = point border + res/2

Parameters

- **points_bounds** (*tuple*) – points' lon_min, lat_min, lon_max, lat_max
- **res** (*float*) – desired resolution in same units as points_bounds
- **crs** (*dict() or rasterio.crs.CRS, optional*) – CRS. Default: DEF_CRS

Returns
centr – Centroids with meta according to given points border data.

Return type
Centroids

set_lat_lon(**args*, ***kwargs*)

This function is deprecated, use Centroids.from_lat_lon instead.

classmethod from_lat_lon(*lat*, *lon*, *crs='EPSG:4326'*)

Create Centroids object from given latitude, longitude and CRS.

Parameters

- **lat** (*np.array*) – latitude
- **lon** (*np.array*) – longitude
- **crs** (*dict() or rasterio.crs.CRS, optional*) – CRS. Default: DEF_CRS

Returns
centr – Centroids with points according to given coordinates

Return type
Centroids

set_raster_file (*file_name*, *band=None*, ***kwargs*)

This function is deprecated, use Centroids.from_raster_file and Centroids.values_from_raster_files instead.

classmethod from_raster_file (*file_name*, *src_crs=None*, *window=None*, *geometry=None*, *dst_crs=None*, *transform=None*, *width=None*, *height=None*, *resampling=Resampling.nearest*)

Create a new Centroids object from a raster file

Select region using window or geometry. Reproject input by providing dst_crs and/or (transform, width, height).

Parameters

- **file_name** (*str*) – path of the file
- **src_crs** (*crs, optional*) – source CRS. Provide it if error without it.
- **window** (*rasterio.windows.Window, optional*) – window to read
- **geometry** (*list of shapely.geometry, optional*) – consider pixels only within these shapes
- **dst_crs** (*crs, optional*) – reproject to given crs
- **transform** (*rasterio.Affine*) – affine transformation to apply
- **wdith** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampling** (*rasterio.warp,.Resampling optional*) – resampling function used for reprojection to dst_crs

Returns
centr – Centroids with meta attribute according to the given raster file

Return type
Centroids

values_from_raster_files (*file_names*, *band=None*, *src_crs=None*, *window=None*, *geometry=None*, *dst_crs=None*, *transform=None*, *width=None*, *height=None*, *resampling=Resampling.nearest*)

Read raster of bands and set 0 values to the masked ones.

Each band is an event. Select region using window or geometry. Reproject input by proving dst_crs and/or (transform, width, height).

Parameters

- **file_names** (*str*) – path of the file
- **band** (*list(int), optional*) – band number to read. Default: [1]
- **src_crs** (*crs, optional*) – source CRS. Provide it if error without it.
- **window** (*rasterio.windows.Window, optional*) – window to read
- **geometry** (*list of shapely.geometry, optional*) – consider pixels only within these shapes
- **dst_crs** (*crs, optional*) – reproject to given crs

- **transform** (*rasterio.Affine*) – affine transformation to apply
- **wdith** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampling** (*rasterio.warp,.Resampling optional*) – resampling function used for reprojection to dst_crs

Raises
`ValueError` –

Returns
inten – Each row is an event.

Return type
scipy.sparse.csr_matrix

`set_vector_file`(*file_name*, *inten_name=None*, ***kwargs*)

This function is deprecated, use Centroids.from_vector_file and Centroids.values_from_vector_files instead.

`classmethod from_vector_file`(*file_name*, *dst_crs=None*)

Create Centroids object from vector file (any format supported by fiona).

Parameters

- **file_name** (*str*) – vector file with format supported by fiona and 'geometry' field.
- **dst_crs** (*crs, optional*) – reproject to given crs

Returns
centr – Centroids with points according to the given vector file

Return type
Centroids

`values_from_vector_files`(*file_names*, *val_names=None*, *dst_crs=None*)

Read intensity or other data from vector files, making sure that geometry is compatible.

If the geometry of the shapes in any of the given files does not agree with the geometry of this Centroids instance, a ValueError is raised.

Parameters

- **file_names** (*list(str)*) – vector files with format supported by fiona and 'geometry' field.
- **val_names** (*list(str), optional*) – list of names of the columns of the values. Default: ['intensity']
- **dst_crs** (*crs, optional*) – reproject to given crs

Raises
`ValueError` –

Returns
values – Sparse array of shape (len(val_name), len(geometry)).

Return type
scipy.sparse.csr_matrix

`read_mat`(**args*, ***kwargs*)

This function is deprecated, use Centroids.from_mat instead.

classmethod from_mat(*file_name*, *var_names=None*)

Read centroids from CLIMADA's MATLAB version.

Parameters

- **file_name** (*str*) – absolute or relative file name
- **var_names** (*dict, optional*) – name of the variables

Raises

`KeyError` –

Returns

centr – Centroids with data from the given file

Return type

Centroids

read_excel(**args*, ***kwargs*)

This function is deprecated, use Centroids.from_excel instead.

classmethod from_excel(*file_name*, *var_names=None*)

Generate a new centroids object from an excel file with column names in var_names.

Parameters

- **file_name** (*str*) – absolute or relative file name
- **var_names** (*dict, default*) – name of the variables

Raises

`KeyError` –

Returns

centr – Centroids with data from the given file

Return type

Centroids

clear()

Clear vector and raster data.

append(*centr*)

Append centroids points.

If centr or self are rasters they are converted to points first using Centroids.set_meta_to_lat_lon. Note that self is modified in-place, and meta is set to {}. Thus, raster information in self is lost.

Note: this is a wrapper for centroids.union.

Parameters

centr (*Centroids*) – Centroids to append. The centroids need to have the same CRS.

See also:

union

Union of Centroid objects.

union(**others*)

Create the union of centroids from the inputs.

The centroids are combined together point by point. Rasters are converted to points and raster information is lost in the output. All centroids must have the same CRS.

In any case, the attribute .geometry is computed for all centroids. This requires a CRS to be defined. If Centroids.crs is None, the default DEF_CRS is set for all centroids (self and others).

When at least one centroids has one of the following property defined, it is also computed for all others. .area_pixel, .dist_coast, .on_land, .region_id, .elevetaion'

!Caution!: the input objects (self and others) are modified in place. Missing properties are added, existing ones are not overwritten.

Parameters
others (*any number of climada.hazard.Centroids()*) – Centroids to form the union with

Returns
centroids – Centroids containing the union of the centroids in others.

Return type
Centroids

Raises
`ValueError` –

`get_closest_point`(*x_lon*, *y_lat*, *scheduler=None*)

Returns closest centroid and its index to a given point.

Parameters

- **x_lon** (*float*) – x coord (lon)
- **y_lat** (*float*) – y coord (lat)
- **scheduler** (*str*) – used for dask map_partitions. "threads", "synchronous" or "processes"

Returns

- **x_close** (*float*) – x-coordinate (longitude) of closest centroid.
- **y_close** (*float*) – y-coordinate (latitude) of closest centroids.
- **idx_close** (*int*) – Index of centroid in internal ordering of centroids.

`set_region_id`(*scheduler=None*)

Set region_id as country ISO numeric code attribute for every pixel or point.

Parameters
scheduler (*str*) – used for dask map_partitions. "threads", "synchronous" or "processes"

`set_area_pixel`(*min_resol=1e-08*, *scheduler=None*)

Set *area_pixel* attribute for every pixel or point (area in m*m).

Parameters

- **min_resol** (*float, optional*) – if centroids are points, use this minimum resolution in lat and lon. Default: 1.0e-8
- **scheduler** (*str*) – used for dask map_partitions. "threads", "synchronous" or "processes"

`set_area_approx`(*min_resol=1e-08*)

Set *area_pixel* attribute for every pixel or point (approximate area in m*m).

Values are differentiated per latitude. Faster than *set_area_pixel*.

Parameters
min_resol (*float, optional*) – if centroids are points, use this minimum resolution in lat and lon. Default: 1.0e-8

set_elevation(*topo_path*)

Set elevation attribute for every pixel or point in meters.

Parameters

topo_path (*str*) – Path to a raster file containing gridded elevation data.

set_dist_coast(*signed=False*, *precomputed=False*, *scheduler=None*)

Set dist_coast attribute for every pixel or point in meters.

Parameters

- **signed** (*bool*) – If True, use signed distances (positive off shore and negative on land). Default: False.
- **precomputed** (*bool*) – If True, use precomputed distances (from NASA). Default: False.
- **scheduler** (*str*) – Used for dask map_partitions. "threads", "synchronous" or "processes"

set_on_land(*scheduler=None*)

Set on_land attribute for every pixel or point.

Parameters

scheduler (*str*) – used for dask map_partitions. "threads", "synchronous" or "processes"

remove_duplicate_points()

Return Centroids with removed duplicated points

Returns

cen – Sub-selection of this object.

Return type

Centroids

select(*reg_id=None*, *extent=None*, *sel_cen=None*)

Return Centroids with points in the given reg_id or within mask

Parameters

- **reg_id** (*int*) – region to filter according to region_id values
- **extent** (*tuple*) – Format (min_lon, max_lon, min_lat, max_lat) tuple. If min_lon > lon_max, the extend crosses the antimeridian and is [lon_max, 180] + [-180, lon_min] Borders are inclusive.
- **sel_cen** (*np.array*) – 1-dim mask, overrides reg_id and extent

Returns

cen – Sub-selection of this object

Return type

Centroids

select_mask(*reg_id=None*, *extent=None*)

Make mask of selected centroids

Parameters

- **reg_id** (*int*) – region to filter according to region_id values
- **extent** (*tuple*) – Format (min_lon, max_lon, min_lat, max_lat) tuple. If min_lon > lon_max, the extend crosses the antimeridian and is [lon_max, 180] + [-180, lon_min] Borders are inclusive.

Returns
sel_cen – 1d mask of selected centroids

Return type
1d array of booleans

set_lat_lon_to_meta(*min_resol=1e-08*)
Compute meta from lat and lon values.

Parameters
min_resol (*float, optional*) – Minimum centroids resolution to use in the raster. Default: 1.0e-8.

set_meta_to_lat_lon()
Compute lat and lon of every pixel center from meta raster.

plot(*axis=None*, *figsize=(9, 13)*, ***kwargs*)
Plot centroids scatter points over earth.

Parameters

- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*(float, float), optional*) – figure size for plt.subplots The default is (9, 13)
- **kwargs** (*optional*) – arguments for scatter matplotlib function

Returns
axis

Return type
matplotlib.axes._subplots.AxesSubplot

calc_pixels_polygons(*scheduler=None*)
Return a gpd.GeoSeries with a polygon for every pixel

Parameters
scheduler (*str*) – used for dask map_partitions. "threads", "synchronous" or "processes"

Returns
geo

Return type
gpd.GeoSeries

empty_geometry_points()
Removes all points in geometry.

Useful when centroids is used in multiprocessing function.

write_hdf5(*file_data*)
Write centroids attributes into hdf5 format.

Parameters
file_data (*str or h5*) – If string, path to write data. If h5 object, the datasets will be generated there.

read_hdf5(**args*, ***kwargs*)
This function is deprecated, use Centroids.from_hdf5 instead.

classmethod from_hdf5(*file_data*)

Create a centroids object from a HDF5 file.

Parameters

file_data (*str or h5*) – If string, path to read data. If h5 object, the datasets will be read from there.

Returns

centr – Centroids with data from the given file

Return type

Centroids

property crs

Get CRS of raster or vector.

property size

Get number of pixels or points.

property shape

Get shape of rastered data.

property total_bounds

Get total bounds (left, bottom, right, top).

property coord

Get [lat, lon] array.

set_geometry_points(*scheduler=None*)

Set *geometry* attribute with Points from *lat/lon* attributes.

Parameters

scheduler (*str*) – used for dask map_partitions. "threads", "synchronous" or "processes"

5.3.2 climada.hazard.base module

class climada.hazard.base.**Hazard**(*haz_type: str = '', pool: ProcessPool | None = None, units: str = '', centroids: Centroids | None = None, event_id: ndarray | None = None, frequency: ndarray | None = None, frequency_unit: str = '1/year', event_name: List[str] | None = None, date: ndarray | None = None, orig: ndarray | None = None, intensity: csr_matrix | None = None, fraction: csr_matrix | None = None*)

Bases: `object`

Contains events of some hazard type defined at centroids. Loads from files with format defined in FILE_EXT.

haz_type

two-letters hazard-type string, e.g., "TC" (tropical cyclone), "RF" (river flood) or "WF" (wild fire). Note: The acronym is used as reference to the hazard when centroids of multiple hazards are assigned to an `Exposures` object.

Type

str

units

units of the intensity

Type
str

centroids
centroids of the events

Type
Centroids

event_id
id (>0) of each event

Type
np.array

event_name
name of each event (default: event_id)

Type
list(str)

date
integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library)

Type
np.array

orig
flags indicating historical events (True) or probabilistic (False)

Type
np.array

frequency
frequency of each event

Type
np.array

frequency_unit
unit of the frequency (default: "1/year")

Type
str

intensity
intensity of the events at centroids

Type
sparse.csr_matrix

fraction
fraction of affected exposures for each event at each centroid. If empty (all 0), it is ignored in the impact computations (i.e., is equivalent to fraction is 1 everywhere).

Type
sparse.csr_matrix

intensity_thres = 10
Intensity threshold per hazard used to filter lower intensities. To be set for every hazard type

vars_oblig = {'centroids', 'event_id', 'fraction', 'frequency', 'intensity', 'units'}

scalar, str, list, 1dim np.array of size num_events, scipy.sparse matrix of shape num_events x num_centroids, Centroids.

Type

Name of the variables needed to compute the impact. Types

vars_def = {'date', 'event_name', 'frequency_unit', 'orig'}

Name of the variables used in impact calculation whose value is descriptive and can therefore be set with default values. Types: scalar, string, list, 1dim np.array of size num_events.

vars_opt = {}

Name of the variables that aren't need to compute the impact. Types: scalar, string, list, 1dim np.array of size num_events.

__init__(*haz_type: str = ''*, *pool: ProcessPool | None = None*, *units: str = ''*, *centroids: Centroids | None = None*, *event_id: ndarray | None = None*, *frequency: ndarray | None = None*, *frequency_unit: str = '1/year'*, *event_name: List[str] | None = None*, *date: ndarray | None = None*, *orig: ndarray | None = None*, *intensity: csr_matrix | None = None*, *fraction: csr_matrix | None = None*)

Initialize values.

Parameters

- **haz_type** (*str, optional*) – acronym of the hazard type (e.g. 'TC').
- **pool** (*pathos.pool, optional*) – Pool that will be used for parallel computation when applicable. Default: None
- **units** (*str, optional*) – units of the intensity. Defaults to empty string.
- **centroids** (*Centroids, optional*) – centroids of the events. Defaults to empty Centroids object.
- **event_id** (*np.array, optional*) – id (>0) of each event. Defaults to empty array.
- **event_name** (*list(str), optional*) – name of each event (default: event_id). Defaults to empty list.
- **date** (*np.array, optional*) – integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library). Defaults to empty array.
- **orig** (*np.array, optional*) – flags indicating historical events (True) or probabilistic (False). Defaults to empty array.
- **frequency** (*np.array, optional*) – frequency of each event. Defaults to empty array.
- **frequency_unit** (*str, optional*) – unit of the frequency (default: "1/year").
- **intensity** (*sparse.csr_matrix, optional*) – intensity of the events at centroids. Defaults to empty matrix.
- **fraction** (*sparse.csr_matrix, optional*) – fraction of affected exposures for each event at each centroid. Defaults to empty matrix.

Examples

Initialize using keyword arguments:

```
>>> haz = Hazard('TC', intensity=sparse.csr_matrix(np.zeros((2, 2))))
```

Take hazard values from file:

```
>>> haz = Hazard.from_mat(HAZ_DEMO_MAT, 'demo')
```

classmethod get_default(*attribute*)

Get the Hazard type default for a given attribute.

Parameters
attribute (*str*) – attribute name

Return type
Any

clear()

Reinitialize attributes (except the process Pool).

check()

Check dimension of attributes.

Raises
ValueError –

classmethod from_raster(*files_intensity*, *files_fraction=None*, *attrs=None*, *band=None*, *haz_type=None*, *pool=None*, *src_crs=None*, *window=None*, *geometry=None*, *dst_crs=None*, *transform=None*, *width=None*, *height=None*, *resampling=Resampling.nearest*)

Create Hazard with intensity and fraction values from raster files

If raster files are masked, the masked values are set to 0.

Files can be partially read using either window or geometry. Additionally, the data is reprojected when custom dst_crs and/or transform, width and height are specified.

Parameters

- **files_intensity** (*list(str)*) – file names containing intensity
- **files_fraction** (*list(str)*) – file names containing fraction
- **attrs** (*dict, optional*) – name of Hazard attributes and their values
- **band** (*list(int), optional*) – bands to read (starting at 1), default [1]
- **haz_type** (*str, optional*) – acronym of the hazard type (e.g. 'TC'). Default: None, which will use the class default (" for vanilla *Hazard* objects, and hard coded in some subclasses)
- **pool** (*pathos.pool, optional*) – Pool that will be used for parallel computation when applicable. Default: None
- **src_crs** (*crs, optional*) – source CRS. Provide it if error without it.
- **window** (*rasterio.windows.Windows, optional*) – window where data is extracted
- **geometry** (*list of shapely.geometry, optional*) – consider pixels only within these shapes
- **dst_crs** (*crs, optional*) – reproject to given crs

- **transform** (*rasterio.Affine*) – affine transformation to apply
- **wdith** (*float, optional*) – number of lons for transform
- **height** (*float, optional*) – number of lats for transform
- **resampling** (*rasterio.warp.Resampling, optional*) – resampling function used for reprojection to dst_crs

Return type
Hazard

set_raster(**args*, ***kwargs*)

This function is deprecated, use Hazard.from_raster.

set_vector(**args*, ***kwargs*)

This function is deprecated, use Hazard.from_vector.

classmethod from_xarray_raster_file(*filepath: Path | str*, **args*, ***kwargs*)

Read raster-like data from a file that can be loaded with xarray

This wraps `from_xarray_raster()` by first opening the target file as xarray dataset and then passing it to that classmethod. Use this wrapper as a simple alternative to opening the file yourself. The signature is exactly the same, except for the first argument, which is replaced by a file path here.

Additional (keyword) arguments are passed to `from_xarray_raster()`.

Parameters
filepath (*Path or str*) – Path of the file to read with xarray. May be any file type supported by xarray. See https://docs.xarray.dev/en/stable/user-guide/io.html

Returns
hazard – A hazard object created from the input data

Return type
climada.Hazard

Examples

```
>>> hazard = Hazard.from_xarray_raster_file("path/to/file.nc", "", "")
```

Notes

If you have specific requirements for opening a data file, prefer opening it yourself and using `from_xarray_raster()`, following this pattern:

```
>>> open_kwargs = dict(engine="h5netcdf", chunks=dict(x=-1, y="auto"))
>>> with xarray.open_dataset("path/to/file.nc", **open_kwargs) as dset:
...     hazard = Hazard.from_xarray_raster(dset, "", "")
```

classmethod from_xarray_raster(*data: Dataset*, *hazard_type: str*, *intensity_unit: str*, ***, *intensity: str = 'intensity'*, *coordinate_vars: Dict[str, str] | None = None*, *data_vars: Dict[str, str] | None = None*, *crs: str = 'EPSG:4326'*, *rechunk: bool = False*)

Read raster-like data from an xarray Dataset

This method reads data that can be interpreted using three coordinates: event, latitude, and longitude. The names of the coordinates to be read from the dataset can be specified via the `coordinate_vars` parameter. The data and the coordinates themselves may be organized in arbitrary dimensions (e.g. two dimensions 'year' and 'altitude' for the coordinate 'event'). See Notes and Examples if you want to load single-event data that does not contain an event dimension.

The only required data is the intensity. For all other data, this method can supply sensible default values. By default, this method will try to find these "optional" data in the Dataset and read it, or use the default values otherwise. Users may specify the variables in the Dataset to be read for certain Hazard object entries, or may indicate that the default values should be used although the Dataset contains appropriate data. This behavior is controlled via the `data_vars` parameter.

If this method succeeds, it will always return a "consistent" Hazard object, meaning that the object can be used in all CLIMADA operations without throwing an error due to missing data or faulty data types.

Use *`from_xarray_raster_file()`* to open a file on disk and load the resulting dataset with this method in one step.

Parameters

- **data** (*xarray.Dataset*) – The dataset to read from.
- **hazard_type** (*str*) – The type identifier of the hazard. Will be stored directly in the hazard object.
- **intensity_unit** (*str*) – The physical units of the intensity.
- **intensity** (*str, optional*) – Identifier of the *xarray.DataArray* containing the hazard intensity data.
- **coordinate_vars** (*dict(str, str), optional*) – Mapping from default coordinate names to coordinate names used in the data to read. The default is `dict(event="time", longitude="longitude", latitude="latitude")`, as most of the commonly used hazard data happens to have a "time" attribute but no "event" attribute.
- **data_vars** (*dict(str, str), optional*) – Mapping from default variable names to variable names used in the data to read. The default names are `fraction`, `hazard_type`, `frequency`, `event_name`, `event_id`, and `date`. If these values are not set, the method tries to load data from the default names. If this fails, the method uses default values for each entry. If the values are set to empty strings (`""`), no data is loaded and the default values are used exclusively. See examples for details.

  Default values are:

  – `date`: The `event` coordinate interpreted as date or ordinal, or ones if that fails (which will issue a warning).

  – `fraction`: `None`, which results in a value of 1.0 everywhere, see *`Hazard.__init__()`* for details.

  – `hazard_type`: Empty string

  – `frequency`: 1.0 for every event

  – `event_name`: String representation of the event date or empty strings if that fails (which will issue a warning).

  – `event_id`: Consecutive integers starting at 1 and increasing with time
- **crs** (*str, optional*) – Identifier for the coordinate reference system of the coordinates. Defaults to `EPSG:4326` (WGS 84), defined by `climada.util.constants.DEF_CRS`. See https://pyproj4.github.io/pyproj/dev/api/crs/crs.html#pyproj.crs.CRS.from_user_input for further information on how to specify the coordinate system.

- **rechunk** (*bool, optional*) – Rechunk the dataset before flattening. This might have serious performance implications. Rechunking in general is expensive, but it might be less expensive than stacking a poorly-chunked array. One event being stored in one chunk would be the optimal configuration. If `rechunk=True`, this will be forced by rechunking the data. Ideally, you would select the chunks in that manner when opening the dataset before passing it to this function. Defaults to `False`.

Returns
hazard – A hazard object created from the input data

Return type
climada.Hazard

See also:

from_xarray_raster_file()
Use this method if you want CLIMADA to open and read a file on disk for you.

Notes

- Single-valued coordinates given by `coordinate_vars`, that are not proper dimensions of the data, are promoted to dimensions automatically. If one of the three coordinates does not exist, use `Dataset.expand_dims` (see https://docs.xarray.dev/en/stable/generated/xarray.Dataset.expand_dims.html and Examples) before loading the Dataset as Hazard.
- Single-valued data for variables `frequency`. `event_name`, and `event_date` will be broadcast to every event.
- The `event` coordinate may take arbitrary values. In case these values cannot be interpreted as dates or date ordinals, the default values for `Hazard.date` and `Hazard.event_name` are used, see the `data_vars`` parameter documentation above.
- To avoid confusion in the call signature, several parameters are keyword-only arguments.
- The attributes `Hazard.haz_type` and `Hazard.unit` currently cannot be read from the Dataset. Use the method parameters to set these attributes.
- This method does not read coordinate system metadata. Use the `crs` parameter to set a custom coordinate system identifier.

Examples

The use of this method is straightforward if the Dataset contains the data with expected names.

```
>>> dset = xr.Dataset(
...     dict(
...         intensity=(
...             ["time", "latitude", "longitude"],
...             [[[0, 1, 2], [3, 4, 5]]],
...         )
...     ),
...     dict(
...         time=[datetime.datetime(2000, 1, 1)],
...         latitude=[0, 1],
...         longitude=[0, 1, 2],
...     ),
```

(continues on next page)

(continued from previous page)

```
... )
>>> hazard = Hazard.from_xarray_raster(dset, "", "")
```

For non-default coordinate names, use the `coordinate_vars` argument.

```
>>> dset = xr.Dataset(
...     dict(
...         intensity=(
...             ["day", "lat", "longitude"],
...             [[[0, 1, 2], [3, 4, 5]]],
...         )
...     ),
...     dict(
...         day=[datetime.datetime(2000, 1, 1)],
...         lat=[0, 1],
...         longitude=[0, 1, 2],
...     ),
... )
>>> hazard = Hazard.from_xarray_raster(
...     dset, "", "", coordinate_vars=dict(event="day", latitude="lat")
... )
```

Coordinates can be different from the actual dataset dimensions. The following loads the data with coordinates `longitude` and `latitude` (default names):

```
>>> dset = xr.Dataset(
...     dict(intensity=(["time", "y", "x"], [[[0, 1, 2], [3, 4, 5]]])),
...     dict(
...         time=[datetime.datetime(2000, 1, 1)],
...         y=[0, 1],
...         x=[0, 1, 2],
...         longitude=(["y", "x"], [[0.0, 0.1, 0.2], [0.0, 0.1, 0.2]]),
...         latitude=(["y", "x"], [[0.0, 0.0, 0.0], [0.1, 0.1, 0.1]]),
...     ),
... )
>>> hazard = Hazard.from_xarray_raster(dset, "", "")
```

Optional data is read from the dataset if the default keys are found. Users can specify custom variables in the data, or that the default keys should be ignored, with the `data_vars` argument.

```
>>> dset = xr.Dataset(
...     dict(
...         intensity=(
...             ["time", "latitude", "longitude"],
...             [[[0, 1, 2], [3, 4, 5]]],
...         ),
...         fraction=(
...             ["time", "latitude", "longitude"],
...             [[[0.0, 0.1, 0.2], [0.3, 0.4, 0.5]]],
...         ),
...         freq=(["time"], [0.4]),
...         event_id=(["time"], [4]),
...     ),
...     dict(
...         time=[datetime.datetime(2000, 1, 1)],
...         latitude=[0, 1],
```

(continues on next page)

(continued from previous page)

```
...         longitude=[0, 1, 2],
...     ),
... )
>>> hazard = Hazard.from_xarray_raster(
...     dset,
...     "",
...     "",
...     data_vars=dict(
...         # Load frequency from 'freq' array
...         frequency="freq",
...         # Ignore 'event_id' array and use default instead
...         event_id="",
...         # 'fraction' array is loaded because it has the default name
...     ),
... )
>>> np.array_equal(hazard.frequency, [0.4]) and np.array_equal(
...     hazard.event_id, [1]
... )
True
```

If your read single-event data your dataset probably will not have a time dimension. As long as a time *coordinate* exists, however, this method will automatically promote it to a dataset dimension and load the data:

```
>>> dset = xr.Dataset(
...     dict(
...         intensity=(
...             ["latitude", "longitude"],
...             [[0, 1, 2], [3, 4, 5]],
...         )
...     ),
...     dict(
...         time=[datetime.datetime(2000, 1, 1)],
...         latitude=[0, 1],
...         longitude=[0, 1, 2],
...     ),
... )
>>> hazard = Hazard.from_xarray_raster(dset, "", "")  # Same as first example
```

If one coordinate is missing altogehter, you must add it or expand the dimensions before loading the dataset:

```
>>> dset = xr.Dataset(
...     dict(
...         intensity=(
...             ["latitude", "longitude"],
...             [[0, 1, 2], [3, 4, 5]],
...         )
...     ),
...     dict(
...         latitude=[0, 1],
...         longitude=[0, 1, 2],
...     ),
... )
>>> dset = dset.expand_dims(time=[numpy.datetime64("2000-01-01")])
>>> hazard = Hazard.from_xarray_raster(dset, "", "")
```

classmethod from_vector(*files_intensity*, *files_fraction=None*, *attrs=None*, *inten_name=None*, *frac_name=None*, *dst_crs=None*, *haz_type=None*)

Read vector files format supported by fiona. Each intensity name is considered an event.

Parameters

- **files_intensity** (*list(str)*) – file names containing intensity, default: ['intensity']
- **files_fraction** (*(list(str))*) – file names containing fraction, default: ['fraction']
- **attrs** (*dict, optional*) – name of Hazard attributes and their values
- **inten_name** (*list(str), optional*) – name of variables containing the intensities of each event
- **frac_name** (*list(str), optional*) – name of variables containing the fractions of each event
- **dst_crs** (*crs, optional*) – reproject to given crs
- **haz_type** (*str, optional*) – acronym of the hazard type (e.g. 'TC'). default: None, which will use the class default (" for vanilla *Hazard* objects, hard coded in some subclasses)

Returns
haz – Hazard from vector file

Return type
climada.hazard.Hazard

reproject_raster(*dst_crs=False*, *transform=None*, *width=None*, *height=None*, *resampl_inten=Resampling.nearest*, *resampl_fract=Resampling.nearest*)

Change current raster data to other CRS and/or transformation

Parameters

- **dst_crs** (*crs, optional*) – reproject to given crs
- **transform** (*rasterio.Affine*) – affine transformation to apply
- **wdith** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampl_inten** (*rasterio.warp,.Resampling optional*) – resampling function used for reprojection to dst_crs for intensity
- **resampl_fract** (*rasterio.warp,.Resampling optional*) – resampling function used for reprojection to dst_crs for fraction

reproject_vector(*dst_crs*, *scheduler=None*)

Change current point data to a a given projection

Parameters

- **dst_crs** (*crs*) – reproject to given crs
- **scheduler** (*str, optional*) – used for dask map_partitions. "threads", "synchronous" or "processes"

raster_to_vector()

Change current raster to points (center of the pixels)

vector_to_raster(*scheduler=None*)

Change current point data to a raster with same resolution

Parameters
scheduler (*str, optional*) – used for dask map_partitions. "threads", "synchronous" or "processes"

read_mat(**args*, ***kwargs*)

This function is deprecated, use Hazard.from_mat.

classmethod from_mat(*file_name*, *var_names=None*)

Read climada hazard generate with the MATLAB code in .mat format.

Parameters

- **file_name** (*str*) – absolute file name
- **var_names** (*dict, optional*) – name of the variables in the file, default: DEF_VAR_MAT constant

Returns
haz – Hazard object from the provided MATLAB file

Return type
climada.hazard.Hazard

Raises
KeyError –

read_excel(**args*, ***kwargs*)

This function is deprecated, use Hazard.from_excel.

classmethod from_excel(*file_name*, *var_names=None*, *haz_type=None*)

Read climada hazard generated with the MATLAB code in Excel format.

Parameters

- **file_name** (*str*) – absolute file name
- **var_names (dict, default)** (*name of the variables in the file,*) – default: DEF_VAR_EXCEL constant
- **haz_type** (*str, optional*) – acronym of the hazard type (e.g. 'TC'). Default: None, which will use the class default (" for vanilla *Hazard* objects, and hard coded in some subclasses)

Returns
haz – Hazard object from the provided Excel file

Return type
climada.hazard.Hazard

Raises
KeyError –

select(*event_names=None*, *event_id=None*, *date=None*, *orig=None*, *reg_id=None*, *extent=None*, *reset_frequency=False*)

Select events matching provided criteria

The frequency of events may need to be recomputed (see *reset_frequency*)!

Parameters

- **event_names** (*list of str, optional*) – Names of events.
- **event_id** (*list of int, optional*) – Id of events. Default is None.
- **date** (*array-like of length 2 containing str or int, optional*) – (initial date, final date) in string ISO format ('2011-01-02') or datetime ordinal integer.
- **orig** (*bool, optional*) – Select only historical (True) or only synthetic (False) events.
- **reg_id** (*int, optional*) – Region identifier of the centroids' region_id attibute.

- **extent** (*tuple(float, float, float, float), optional*) – Extent of centroids as (min_lon, max_lon, min_lat, max_lat). The default is None.
- **reset_frequency** (*bool, optional*) – Change frequency of events proportional to difference between first and last year (old and new). Default: False.

Returns
haz – If no event matching the specified criteria is found, None is returned.

Return type
Hazard or None

select_tight(*buffer=0.8999280057595392*, *val='intensity'*)

Reduce hazard to those centroids spanning a minimal box which contains all non-zero intensity or fraction points.

Parameters

- **buffer** (*float, optional*) – Buffer of box in the units of the centroids. The default is approximately equal to the default threshold from the assign_centroids method (works if centroids in lat/lon)
- **val** (*string, optional*) – Select tight by non-zero 'intensity' or 'fraction'. The default is 'intensity'.

Returns
Copy of the Hazard with centroids reduced to minimal box. All other hazard properties are carried over without changes.

Return type
Hazard

See also:

self.select
Method to select centroids by lat/lon extent

util.coordinates.match_coordinates
algorithm to match centroids.

local_exceedance_inten(*return_periods=(25, 50, 100, 250)*)

Compute exceedance intensity map for given return periods.

Parameters
return_periods (*np.array*) – return periods to consider

Returns
inten_stats

Return type
np.array

plot_rp_intensity(*return_periods=(25, 50, 100, 250)*, *smooth=True*, *axis=None*, *figsize=(9, 13)*, *adapt_fontsize=True*, ***kwargs*)

Compute and plot hazard exceedance intensity maps for different return periods. Calls local_exceedance_inten.

Parameters

- **return_periods** (*tuple(int), optional*) – return periods to consider
- **smooth** (*bool, optional*) – smooth plot to plot.RESOLUTIONxplot.RESOLUTION

- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*tuple, optional*) – figure size for plt.subplots
- **kwargs** (*optional*) – arguments for pcolormesh matplotlib function used in event plots

Returns
axis, inten_stats – intenstats is return_periods.size x num_centroids

Return type
matplotlib.axes._subplots.AxesSubplot, np.ndarray

plot_intensity(*event=None*, *centr=None*, *smooth=True*, *axis=None*, *adapt_fontsize=True*, ***kwargs*)

Plot intensity values for a selected event or centroid.

Parameters

- **event** (*int or str, optional*) – If event > 0, plot intensities of event with id = event. If event = 0, plot maximum intensity in each centroid. If event < 0, plot abs(event)-largest event. If event is string, plot events with that name.
- **centr** (*int or tuple, optional*) – If centr > 0, plot intensity of all events at centroid with id = centr. If centr = 0, plot maximum intensity of each event. If centr < 0, plot abs(centr)-largest centroid where higher intensities are reached. If tuple with (lat, lon) plot intensity of nearest centroid.
- **smooth** (*bool, optional*) – Rescale data to RESOLUTIONxRESOLUTION pixels (see constant in module *climada.util.plot*)
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for pcolormesh matplotlib function used in event plots or for plot function used in centroids plots

Return type
matplotlib.axes._subplots.AxesSubplot

Raises
`ValueError` –

plot_fraction(*event=None*, *centr=None*, *smooth=True*, *axis=None*, ***kwargs*)

Plot fraction values for a selected event or centroid.

Parameters

- **event** (*int or str, optional*) – If event > 0, plot fraction of event with id = event. If event = 0, plot maximum fraction in each centroid. If event < 0, plot abs(event)-largest event. If event is string, plot events with that name.
- **centr** (*int or tuple, optional*) – If centr > 0, plot fraction of all events at centroid with id = centr. If centr = 0, plot maximum fraction of each event. If centr < 0, plot abs(centr)-largest centroid where highest fractions are reached. If tuple with (lat, lon) plot fraction of nearest centroid.
- **smooth** (*bool, optional*) – Rescale data to RESOLUTIONxRESOLUTION pixels (see constant in module *climada.util.plot*)
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for pcolormesh matplotlib function used in event plots or for plot function used in centroids plots

Return type
matplotlib.axes._subplots.AxesSubplot

Raises
`ValueError` –

sanitize_event_ids()

Make sure that event ids are unique

get_event_id(*event_name*)

Get an event id from its name. Several events might have the same name.

Parameters
event_name (*str*) – Event name

Returns
list_id

Return type
np.array(int)

get_event_name(*event_id*)

Get the name of an event id.

Parameters
event_id (*int*) – id of the event

Return type
str

Raises
`ValueError` –

get_event_date(*event=None*)

Return list of date strings for given event or for all events, if no event provided.

Parameters
event (*str or int, optional*) – event name or id.

Returns
l_dates

Return type
list(str)

calc_year_set()

From the dates of the original events, get number yearly events.

Returns
orig_yearset – key are years, values array with event_ids of that year

Return type
dict

remove_duplicates()

Remove duplicate events (events with same name and date).

set_frequency(*yearrange=None*)

Set hazard frequency from yearrange or intensity matrix.

Parameters
yearrange (*tuple or list, optional*) – year range to be used to compute frequency per event. If yearrange is not given (None), the year range is derived from self.date

property size

Return number of events.

write_raster (*file_name*, *intensity=True*)

Write intensity or fraction as GeoTIFF file. Each band is an event

Parameters

- **file_name** (*str*) – file name to write in tif format
- **intensity** (*bool*) – if True, write intensity, otherwise write fraction

write_hdf5 (*file_name*, *todense=False*)

Write hazard in hdf5 format.

Parameters

- **file_name** (*str*) – file name to write, with h5 format
- **todense** (*bool*) – if True write the sparse matrices as hdf5.dataset by converting them to dense format first. This increases readability of the file for other programs. default: False

read_hdf5 (**args*, ***kwargs*)

This function is deprecated, use Hazard.from_hdf5.

classmethod from_hdf5 (*file_name*)

Read hazard in hdf5 format.

Parameters

file_name (*str*) – file name to read, with h5 format

Returns

haz – Hazard object from the provided MATLAB file

Return type

climada.hazard.Hazard

append (**others*)

Append the events and centroids to this hazard object.

All of the given hazards must be of the same type and use the same units as self. The centroids of all hazards must have the same CRS.

The following kinds of object attributes are processed:

- All centroids are combined together using *Centroids.union*.
- Lists, 1-dimensional arrays (NumPy) and sparse CSR matrices (SciPy) are concatenated. Sparse matrices are concatenated along the first (vertical) axis.

For any other type of attribute: A ValueError is raised if an attribute of that name is not defined in all of the non-empty hazards at least. However, there is no check that the attribute value is identical among the given hazard objects. The initial attribute value of *self* will not be modified.

Note: Each of the hazard's *centroids* attributes might be modified in place in the sense that missing properties are added, but existing ones are not overwritten. In case of raster centroids, conversion to point centroids is applied so that raster information (meta) is lost. For more information, see *Centroids.union*.

Parameters

others (*one or more climada.hazard.Hazard objects*) – Hazard instances to append to self

Raises

TypeError, ValueError –

See also:

Hazard.concat
concatenate 2 or more hazards

Centroids.union
combine centroids

classmethod concat(*haz_list*)

Concatenate events of several hazards of same type.

This function creates a new hazard of the same class as the first hazard in the given list and then applies the *append* method. Please refer to the docs of *Hazard.append* for caveats and limitations of the concatenation procedure.

For centroids, lists, arrays and sparse matrices, the remarks in *Hazard.append* apply. All other attributes are copied from the first object in *haz_list*.

Note that *Hazard.concat* can be used to concatenate hazards of a subclass. The result's type will be the subclass. However, calling *concat([])* (with an empty list) is equivalent to instantiation without init parameters. So, *Hazard.concat([])* is equivalent to *Hazard()*. If *HazardB* is a subclass of *Hazard*, then *HazardB.concat([])* is equivalent to *HazardB()* (unless *HazardB* overrides the *concat* method).

Parameters
haz_list (*list of climada.hazard.Hazard objects*) – Hazard instances of the same hazard type (subclass).

Returns
haz_concat – This will be of the same type (subclass) as all the hazards in *haz_list*.

Return type
instance of climada.hazard.Hazard

See also:

Hazard.append
append hazards to a hazard in place

Centroids.union
combine centroids

change_centroids(*centroids*, *threshold=100*)

Assign (new) centroids to hazard.

Centoids of the hazard not in centroids are mapped onto the nearest point. Fails if a point is further than threshold from the closest centroid.

The centroids must have the same CRS as self.centroids.

Parameters

- **haz** (*Hazard*) – Hazard instance
- **centroids** (*Centroids*) – Centroids instance on which to map the hazard.
- **threshold** (*int or float*) – Threshold (in km) for mapping haz.centroids not in centroids. Argument is passed to climada.util.coordinates.match_coordinates. Default: 100 (km)

Returns
haz_new_cent – Hazard projected onto centroids

Return type
Hazard

Raises
ValueError –

See also:

util.coordinates.match_coordinates
algorithm to match centroids.

property centr_exp_col
Name of the centroids columns for this hazard in an exposures

Returns
centroids string indicator with hazard type defining column in an exposures gdf. E.g. "centr_TC"

Return type
String

get_mdr(*cent_idx*, *impf*)
Return Mean Damage Ratio (mdr) for chosen centroids (cent_idx) for given impact function.

Parameters

- **cent_idx** (*array-like*) – array of indices of chosen centroids from hazard
- **impf** (*ImpactFunc*) – impact function to compute mdr

Returns
sparse matrix (n_events x len(cent_idx)) with mdr values

Return type
sparse.csr_matrix

See also:

get_paa
get the paa ffor the given centroids

get_paa(*cent_idx*, *impf*)
Return Percentage of Affected Assets (paa) for chosen centroids (cent_idx) for given impact function.

Note that value as intensity = 0 are ignored. This is different from get_mdr.

Parameters

- **cent_idx** (*array-like*) – array of indices of chosen centroids from hazard
- **impf** (*ImpactFunc*) – impact function to compute mdr

Returns
sparse matrix (n_events x len(cent_idx)) with paa values

Return type
sparse.csr_matrix

See also:

get_mdr
get the mean-damage ratio for the given centroids

5.3.3 climada.hazard.isimip_data module

5.3.4 climada.hazard.storm_europe module

class climada.hazard.storm_europe.**StormEurope**(*units: str = 'm/s'*, *ssi: ndarray | None = None*, *ssi_wisc: ndarray | None = None*, *ssi_full_area: ndarray | None = None*, ***kwargs*)

Bases: *Hazard*

A hazard set containing european winter storm events. Historic storm events can be downloaded at http://wisc.climate.copernicus.eu/ and read with *from_footprints*. Weather forecasts can be automatically downloaded from https://opendata.dwd.de/ and read with from_icon_grib(). Weather forecast from the COSMO-Consortium http://www.cosmo-model.org/ can be read with from_cosmoe_file().

ssi_wisc

Storm Severity Index (SSI) as recorded in the footprint files; apparently not reproducible from the footprint values only.

Type

np.array, float

ssi

SSI as set by set_ssi; uses the Dawkins definition by default.

Type

np.array, float

intensity_thres = 14.7

Intensity threshold for storage in m/s; same as used by WISC SSI calculations.

vars_opt = {'ssi', 'ssi_full_area', 'ssi_wisc'}

Name of the variables that aren't need to compute the impact.

__init__(*units: str = 'm/s'*, *ssi: ndarray | None = None*, *ssi_wisc: ndarray | None = None*, *ssi_full_area: ndarray | None = None*, ***kwargs*)

Initialize a StormEurope object

Parameters

- **units** (*str*) – The units of the Hazard intensity. Defaults to 'm/s'
- **ssi** (*numpy.ndarray*) – The Storm Severity Index (SSI). 1d vector, same length as number of storms. Defaults to an empty array.
- **ssi_wisc** (*numpy.ndarray*) – The Storm Severity Index (SSI) as reconded in the footprint files. 1d vector, same length as number of storms. Defaults to an empty array.
- **ssi_full_area** (*numpy.ndarray*) – 1d vector, same length as number of storms. Used to conserve the the index that is derived from the area and windspeed of the full geographic extent of a storm system. Needed for cropping the *StormEurope* object into a smaller region`. Defaults to an empty array.

read_footprints(**args*, ***kwargs*)

This function is deprecated, use StormEurope.from_footprints instead.

classmethod from_footprints(*path*, *ref_raster=None*, *centroids=None*, *files_omit='fp_era20c_1990012515_701_0.nc'*, *combine_threshold=None*, *intensity_thres=None*)

Create new StormEurope object from WISC footprints.

Assumes that all footprints have the same coordinates as the first file listed/first file in dir.

Parameters

- **path** (*str, list(str)*) – A location in the filesystem. Either a path to a single netCDF WISC footprint, or a folder containing only footprints, or a globbing pattern to one or more footprints.
- **ref_raster** (*str, optional*) – Reference netCDF file from which to construct a new barebones Centroids instance. Defaults to the first file in path.
- **centroids** (*Centroids, optional*) – A Centroids struct, overriding ref_raster
- **files_omit** (*str, list(str), optional*) – List of files to omit; defaults to one duplicate storm present in the WISC set as of 2018-09-10.
- **combine_threshold** (*int, optional*) – threshold for combining events in number of days. if the difference of the dates (self.date) of two events is smaller or equal to this threshold, the two events are combined into one. Default is None, Advised for WISC is 2
- **intensity_thres** (*float, optional*) – Intensity threshold for storage in m/s. Default: class attribute StormEurope.intensity_thres (same as used by WISC SSI calculations)

Returns
haz – StormEurope object with data from WISC footprints.

Return type
StormEurope

read_cosmoe_file(**args*, ***kwargs*)

This function is deprecated, use StormEurope.from_cosmoe_file instead.

classmethod from_cosmoe_file(*fp_file*, *run_datetime*, *event_date=None*, *model_name='COSMO-2E'*, *description=None*, *intensity_thres=None*)

Create a new StormEurope object with gust footprint from weather forecast.

The funciton is designed for the COSMO ensemble model used by the COSMO Consortium http://www.cosmo-model.org/ and postprocessed to an netcdf file using fieldextra. One event is one full day in UTC. Works for MeteoSwiss model output of COSMO-1E (11 members, resolution 1.1 km, forecast period 33-45 hours) COSMO-2E (21 members, resolution 2.2 km, forecast period 5 days)

The frequency of each event is informed by their probability in the ensemble forecast and is equal to 1/11 or 1/21 for COSMO-1E or COSMO-2E, respectively.

Parameters

- **fp_file** (*str*) – string directing to one netcdf file
- **run_datetime** (*datetime*) – The starting timepoint of the forecast run of the cosmo model
- **event_date** (*datetime, optional*) – one day within the forecast period, only this day (00H-24H) will be included in the hazard
- **model_name** (*str,optional*) – provide the name of the COSMO model, for the description (e.g., 'COSMO-1E', 'COSMO-2E')
- **description** (*str, optional*) – description of the events, defaults to a combination of model_name and run_datetime
- **intensity_thres** (*float, optional*) – Intensity threshold for storage in m/s. Default: class attribute StormEurope.intensity_thres (same as used by WISC SSI calculations)

Returns
haz – StormEurope object with data from COSMO ensemble file.

Return type
StormEurope

`read_icon_grib`(**args*, ***kwargs*)

This function is deprecated, use StormEurope.from_icon_grib instead.

`classmethod from_icon_grib`(*run_datetime*, *event_date=None*, *model_name='icon-eu-eps'*, *description=None*, *grib_dir=None*, *delete_raw_data=True*, *intensity_thres=None*)

Create new StormEurope object from DWD icon weather forecast footprints.

New files are available for 24 hours on https://opendata.dwd.de, old files can be processed if they are already stored in grib_dir. One event is one full day in UTC. Current setup works for runs starting at 00H and 12H. Otherwise the aggregation is inaccurate, because of the given file structure with 1-hour, 3-hour and 6-hour maxima provided.

The frequency for one event is 1/(number of ensemble members)

Parameters

- **run_datetime** (*datetime*) – The starting timepoint of the forecast run of the icon model
- **event_date** (*datetime, optional*) – one day within the forecast period, only this day (00H-24H) will be included in the hazard
- **model_name** (*str,optional*) – select the name of the icon model to be downloaded. Must match the url on https://opendata.dwd.de (see download_icon_grib for further info)
- **description** (*str, optional*) – description of the events, defaults to a combination of model_name and run_datetime
- **grib_dir** (*str, optional*) – path to folder, where grib files are or should be stored
- **delete_raw_data** (*bool,optional*) – select if downloaded raw data in .grib.bz2 file format should be stored on the computer or removed
- **intensity_thres** (*float, optional*) – Intensity threshold for storage in m/s. Default: class attribute StormEurope.intensity_thres (same as used by WISC SSI calculations)

Returns
haz – StormEurope object with data from DWD icon weather forecast footprints.

Return type
StormEurope

`calc_ssi`(*method='dawkins'*, *intensity=None*, *on_land=True*, *threshold=None*, *sel_cen=None*)

Calculate the SSI, method must either be 'dawkins' or 'wisc_gust'.

'dawkins', after Dawkins et al. (2016), doi:10.5194/nhess-16-1999-2016, matches the MATLAB version. ssi = sum_i(area_cell_i * intensity_cell_i^3)

'wisc_gust', according to the WISC Tier 1 definition found at https://wisc.climate.copernicus.eu/wisc/#/help/products#tier1_section ssi = sum(area_on_land) * mean(intensity)^3

In both definitions, only raster cells that are above the threshold are used in the computation. Note that this method does not reproduce self.ssi_wisc, presumably because the footprint only contains the maximum wind gusts instead of the sustained wind speeds over the 72 hour window. The deviation may also be due to differing definitions of what lies on land (i.e. Syria, Russia, Northern Africa and Greenland are exempt).

Parameters

- **method** (*str*) – Either 'dawkins' or 'wisc_gust'
- **intensity** (*scipy.sparse.csr*) – Intensity matrix; defaults to self.intensity
- **on_land** (*bool*) – Only calculate the SSI for areas on land, ignoring the intensities at sea. Defaults to true, whereas the MATLAB version did not.
- **threshold** (*float, optional*) – Intensity threshold used in index definition. Cannot be lower than the read-in value.
- **sel_cen** (*np.array, bool*) – A boolean vector selecting centroids. Takes precendence over on_land.

set_ssi(***kwargs*)

Wrapper around calc_ssi for setting the self.ssi attribute.

Parameters
kwargs – passed on to calc_ssi

plot_ssi(*full_area=False*)

Plot the distribution of SSIs versus their cumulative exceedance
frequencies, highlighting historical storms in red.

Returns

- **fig** (*matplotlib.figure.Figure*)
- **ax** (*matplotlib.axes._subplots.AxesSubplot*)

generate_prob_storms(*reg_id=528*, *spatial_shift=4*, *ssi_args=None*, ***kwargs*)

Generates a new hazard set with one original and 29 probabilistic storms per historic storm. This represents a partial implementation of the Monte-Carlo method described in section 2.2 of Schwierz et al. (2010), doi:10.1007/s10584-009-9712-1. It omits the rotation of the storm footprints, as well as the pseudo- random alterations to the intensity.

In a first step, the original intensity and five additional intensities are saved to an array. In a second step, those 6 possible intensity levels are shifted by n raster pixels into each direction (N/S/E/W).

Caveats:

- Memory safety is an issue; trial with the entire dataset resulted in 60GB of swap memory being used…
- Can only use numeric region_id for country selection
- Drops event names as provided by WISC

Parameters

- **region_id** (*int, list of ints, or None*) – iso_n3 code of the countries we want the generated hazard set to be returned for.
- **spatial_shift** (*int*) – amount of raster pixels to shift by
- **ssi_args** (*dict*) – A dictionary of arguments passed to calc_ssi
- **kwargs** – keyword arguments passed on to self._hist2prob()

Returns
new_haz – A new hazard set for the given country. Centroid attributes are preserved. self.orig attribute is set to True for original storms (event_id ending in 00). Also contains a ssi_prob attribute,

Return type
StormEurope

5.3.5 climada.hazard.tag module

5.3.6 climada.hazard.tc_clim_change module

`climada.hazard.tc_clim_change.`**`TOT_RADIATIVE_FORCE = PosixPath('/home/docs/climada/data/rcp_db.xls')`**

//www.iiasa.ac.at/web-apps/tnt/RcpDb. generated: 2018-07-04 10:47:59.

Type
© RCP Database (Version 2.0.5) http

`climada.hazard.tc_clim_change.`**`get_knutson_criterion`**`()`

Fill changes in TCs according to Knutson et al. 2015 Global projections of intense tropical cyclone activity for the late twenty-first century from dynamical downscaling of CMIP5/RCP4.5 scenarios.

Returns
criterion – list of the criterion dictionary for frequency and intensity change per basin, per category taken from the Table 3 in Knutson et al. 2015. with items 'basin' (str), 'category' (list(int)), 'year' (int), 'change' (float), 'variable' ('intensity' or 'frequency')

Return type
list(dict)

`climada.hazard.tc_clim_change.`**`calc_scale_knutson`**`(`*ref_year=2050*, *rcp_scenario=45*`)`

Comparison 2081-2100 (i.e., late twenty-first century) and 2001-20 (i.e., present day). Late twenty-first century effects on intensity and frequency per Saffir-Simpson-category and ocean basin is scaled to target year and target RCP proportional to total radiative forcing of the respective RCP and year.

Parameters

- **ref_year** (*int, optional*) – year between 2000 ad 2100. Default: 2050
- **rcp_scenario** (*int, optional*) – 26 for RCP 2.6, 45 for RCP 4.5. The default is 45 60 for RCP 6.0 and 85 for RCP 8.5.

Returns
factor – factor to scale Knutson parameters to the give RCP and year

Return type
float

5.3.7 climada.hazard.tc_tracks module

`climada.hazard.tc_tracks.`**`CAT_NAMES = {-1: 'Tropical Depression', 0: 'Tropical Storm', 1: 'Hurricane Cat. 1', 2: 'Hurricane Cat. 2', 3: 'Hurricane Cat. 3', 4: 'Hurricane Cat. 4', 5: 'Hurricane Cat. 5'}`**

Saffir-Simpson category names.

`climada.hazard.tc_tracks.`**`SAFFIR_SIM_CAT = [34, 64, 83, 96, 113, 137, 1000]`**

Saffir-Simpson Hurricane Wind Scale in kn based on NOAA

class `climada.hazard.tc_tracks.`**`TCTracks`**(*data: List[Dataset] | None = None*, *pool: ProcessPool | None = None*)

Bases: `object`

Contains tropical cyclone tracks.

`data`

List of tropical cyclone tracks. Each track contains following attributes:

- time (coords)
- lat (coords)
- lon (coords)
- time_step (in hours)
- radius_max_wind (in nautical miles)
- radius_oci (in nautical miles)
- max_sustained_wind (in knots)
- central_pressure (in hPa/mbar)
- environmental_pressure (in hPa/mbar)
- basin (for each track position)
- max_sustained_wind_unit (attrs)
- central_pressure_unit (attrs)
- name (attrs)
- sid (attrs)
- orig_event_flag (attrs)
- data_provider (attrs)
- id_no (attrs)
- category (attrs)

Computed during processing:

- on_land (bool for each track position)
- dist_since_lf (in km)

Additional data variables such as "nature" (specifiying, for each track position, whether a system is a disturbance, tropical storm, post-transition extratropical storm etc.) might be included, depending on the data source and on use cases.

Type

list(xarray.Dataset)

`__init__`(*data: List[Dataset] | None = None*, *pool: ProcessPool | None = None*)

Create new (empty) TCTracks instance.

Parameters

- **data** (*list of xarray.Dataset, optional*) – List of tropical cyclone tracks, each stored as single xarray Dataset. See the Attributes for a full description of the required Dataset variables and attributes. Defaults to an empty list.

- **pool** (*pathos.pools, optional*) – Pool that will be used for parallel computation when applicable. Default: None

append (*tracks*)

Append tracks to current.

Parameters
tracks (*xarray.Dataset or list(xarray.Dataset)*) – tracks to append.

get_track (*track_name=None*)

Get track with provided name.

Returns the first matching track based on the assumption that no other track with the same name or sid exists in the set.

Parameters
track_name (*str, optional*) – Name or sid (ibtracsID for IBTrACS) of track. If None (default), return all tracks.

Returns
result – Usually, a single track is returned. If no track with the specified name is found, an empty list *[]* is returned. If called with *track_name=None*, the list of all tracks is returned.

Return type
xarray.Dataset or list of xarray.Dataset

subset (*filterdict*)

Subset tracks based on track attributes.

Select all tracks matching exactly the given attribute values.

Parameters
filterdict (*dict or OrderedDict*) – Keys are attribute names, values are the corresponding attribute values to match. In case of an ordered dict, the filters are applied in the given order.

Returns
tc_tracks – A new instance of TCTracks containing only the matching tracks.

Return type
TCTracks

tracks_in_exp (*exposure*, *buffer=1.0*)

Select only the tracks that are in the vicinity (buffer) of an exposure.

Each exposure point/geometry is extended to a disc of radius *buffer*. Each track is converted to a line and extended by a radius *buffer*.

Parameters

- **exposure** (*Exposure*) – Exposure used to select tracks.
- **buffer** (*float, optional*) – Size of buffer around exposure geometries (in the units of *exposure.crs*), see *geopandas.distance*. Default: 1.0

Returns
filtered_tracks – TCTracks object with tracks from tc_tracks intersecting the exposure whitin a buffer distance.

Return type
TCTracks

read_ibtracs_netcdf(**args*, ***kwargs*)

This function is deprecated, use TCTracks.from_ibtracs_netcdf instead.

classmethod from_ibtracs_netcdf(*provider=None*, *rescale_windspeeds=True*, *storm_id=None*, *year_range=None*, *basin=None*, *genesis_basin=None*, *interpolate_missing=True*, *estimate_missing=False*, *correct_pres=False*, *discard_single_points=True*, *additional_variables=None*, *file_name='IBTrACS.ALL.v04r00.nc'*)

Create new TCTracks object from IBTrACS databse.

When using data from IBTrACS, make sure to be familiar with the scope and limitations of IBTrACS, e.g. by reading the official documentation (https://www.ncdc.noaa.gov/ibtracs/pdf/IBTrACS_version4_Technical_Details.pdf). Reading the CLIMADA documentation can't replace a thorough understanding of the underlying data. This function only provides a (hopefully useful) interface for the data input, but cannot provide any guidance or make recommendations about if and how to use IBTrACS data for your particular project.

Resulting tracks are required to have both pressure and wind speed information at all time steps. Therefore, all track positions where one of wind speed or pressure are missing are discarded unless one of *interpolate_missing* or *estimate_missing* are active.

Some corrections are automatically applied, such as: *environmental_pressure* is enforced to be larger than *central_pressure*.

Note that the tracks returned by this function might contain irregular time steps since that is often the case for the original IBTrACS records: many agencies add an additional time step at landfall. Apply the *equal_timestep* function afterwards to enforce regular time steps.

Parameters

- **provider** (*str or list of str, optional*) – Either specify an agency, such as "usa", "newdelhi", "bom", "cma", "tokyo", or the special values "official" and "official_3h":
  - "official" means using the (usually 6-hourly) officially reported values of the officially responsible agencies.
  - "official_3h" means to include (inofficial) 3-hourly data of the officially responsible agencies (whenever available).

  If you want to restrict to the officially reported values by the officially responsible agencies (*provider="official"*) without any modifications to the original official data, make sure to also set *estimate_missing=False* and *interpolate_missing=False*. Otherwise, gaps in the official reporting will be filled using interpolation and/or statistical estimation procedures (see below). If a list is given, the following logic is applied: For each storm, the variables that are not reported by the first agency for this storm are taken from the next agency in the list that did report this variable for this storm. For different storms, the same variable might be taken from different agencies. Default: `['official_3h', 'usa', 'tokyo', 'newdelhi', 'reunion', 'bom', 'nadi', 'wellington', 'cma', 'hko', 'ds824', 'td9636', 'td9635', 'neumann', 'mlc']`
- **rescale_windspeeds** (*bool, optional*) – If True, all wind speeds are linearly rescaled to 1-minute sustained winds. Note however that the IBTrACS documentation (Section 5.2, https://www.ncdc.noaa.gov/ibtracs/pdf/IBTrACS_version4_Technical_Details.pdf) includes a warning about this kind of conversion: "While a multiplicative factor can describe the numerical differences, there are procedural and observational differences between agencies that can change through time, which confounds the simple multiplicative factor." Default: True
- **storm_id** (*str or list of str, optional*) – IBTrACS ID of the storm, e.g. 1988234N13299, [1988234N13299, 1989260N11316].

- **year_range** (*tuple (min_year, max_year), optional*) – Year range to filter track selection. Default: None.
- **basin** (*str, optional*) – If given, select storms that have at least one position in the specified basin. This allows analysis of a given basin, but also means that basin-specific track sets should not be combined across basins since some storms will be in more than one set. If you would like to select storms by their (unique) genesis basin instead, use the parameter *genesis_basin*. For possible values (basin abbreviations), see the parameter *genesis_basin*. If None, this filter is not applied. Default: None.
- **genesis_basin** (*str, optional*) – The basin where a TC is formed is not defined in IBTrACS. However, this filter option allows to restrict to storms whose first valid eye position is in the specified basin, which simulates the genesis location. Note that the resulting genesis basin of a particular track may depend on the selected *provider* and on *estimate_missing* because only the first *valid* eye position is considered. Possible values are 'NA' (North Atlantic), 'SA' (South Atlantic), 'EP' (Eastern North Pacific, which includes the Central Pacific region), 'WP' (Western North Pacific), 'SP' (South Pacific), 'SI' (South Indian), 'NI' (North Indian). If None, this filter is not applied. Default: None.
- **interpolate_missing** (*bool, optional*) – If True, interpolate temporal reporting gaps within a variable (such as pressure, wind speed, or radius) linearly if possible. Temporal interpolation is with respect to the time steps defined in IBTrACS for a particular storm. No new time steps are added that are not originally defined in IBTrACS. For each time step with a missing value, this procedure is only able to fill in that value if there are other time steps before and after this time step for which values have been reported. This procedure will be applied before the statistical estimations referred to by *estimate_missing*. It is applied to all variables (eye position, wind speed, environmental and central pressure, storm radius and radius of maximum winds). Default: True
- **estimate_missing** (*bool, optional*) – For each fixed time step, estimate missing pressure, wind speed and radius using other variables that are available at that time step. The relationships between the variables are purely statistical. In comparison to *interpolate_missing*, this procedure is able to estimate values for variables that haven't been reported by any agency at any time step, as long as other variables are available. A typical example are storms before 1950, for which there are often no reported values for pressure, but for wind speed. In this case, a rough statistical pressure-wind relationship is applied to estimate the missing pressure values from the available wind-speed values. Make sure to set *rescale_windspeeds=True* when using this option because the statistical relationships are calibrated using rescaled wind speeds. Default: False
- **correct_pres** (*bool, optional*) – For backwards compatibility, alias for *estimate_missing*. This is deprecated, use *estimate_missing* instead!
- **discard_single_points** (*bool, optional*) – Whether to discard tracks that consists of a single point. Recommended for full compatiblity with other functions such as *equal_timesteps*. Default: True.
- **file_name** (*str, optional*) – Name of NetCDF file to be dowloaded or located at climada/data/system. Default: 'IBTrACS.ALL.v04r00.nc'
- **additional_variables** (*list of str, optional*) – If specified, additional IBTrACS data variables are extracted, such as "nature" or "storm_speed". Only variables that are not agency-specific are supported. Default: None.

Returns
tracks – TCTracks with data from IBTrACS

Return type
TCTracks

read_processed_ibtracs_csv(**args*, ***kwargs*)

This function is deprecated, use TCTracks.from_processed_ibtracs_csv instead.

classmethod from_processed_ibtracs_csv(*file_names*)

Create TCTracks object from processed ibtracs CSV file(s).

Parameters
file_names (*str or list of str*) – Absolute file name(s) or folder name containing the files to read.

Returns
tracks – TCTracks with data from the processed ibtracs CSV file.

Return type
TCTracks

read_simulations_emanuel(**args*, ***kwargs*)

This function is deprecated, use TCTracks.from_simulations_emanuel instead.

classmethod from_simulations_emanuel(*file_names*, *hemisphere=None*, *subset=None*)

Create new TCTracks object from Kerry Emanuel's tracks.

Parameters

- **file_names** (*str or list of str*) – Absolute file name(s) or folder name containing the files to read.
- **hemisphere** (*str or None, optional*) – For global data sets, restrict to northern ('N') or southern ('S') hemisphere. Default: None (no restriction)
- **subset** (*list of int, optional*) – If given, only include the tracks with the given indices. Since the simulation files can be huge, this feature is useful for running tests on smaller subsets or on random subsamples. Default: None

Returns
tracks – TCTracks with data from Kerry Emanuel's simulations.

Return type
TCTracks

read_one_gettelman(*nc_data*, *i_track*)

This function is deprecated, use TCTracks.from_gettelman instead.

classmethod from_gettelman(*path*)

Create new TCTracks object from Andrew Gettelman's tracks.

Parameters
path (*str or Path*) – Path to one of Andrew Gettelman's NetCDF files.

Returns
tracks – TCTracks with data from Andrew Gettelman's simulations.

Return type
TCTracks

read_simulations_chaz(**args*, ***kwargs*)

This function is deprecated, use TCTracks.from_simulations_chaz instead.

classmethod from_simulations_chaz(*file_names*, *year_range=None*, *ensemble_nums=None*)

Create new TCTracks object from CHAZ simulations

Lee, C.-Y., Tippett, M.K., Sobel, A.H., Camargo, S.J. (2018): An Environmentally Forced Tropical Cyclone Hazard Model. J Adv Model Earth Sy 10(1): 223–241.

Parameters

- **file_names** (*str or list of str*) – Absolute file name(s) or folder name containing the files to read.
- **year_range** (*tuple (min_year, max_year), optional*) – Filter by year, if given.
- **ensemble_nums** (*list, optional*) – Filter by ensembleNum, if given.

Returns
tracks – TCTracks with data from the CHAZ simulations.

Return type
TCTracks

read_simulations_storm(**args*, ***kwargs*)

This function is deprecated, use TCTracks.from_simulations_storm instead.

classmethod from_simulations_storm(*path*, *years=None*)

Create new TCTracks object from STORM simulations

Bloemendaal et al. (2020): Generation of a global synthetic tropical cyclone hazard dataset using STORM. Scientific Data 7(1): 40.

Track data available for download from

https://doi.org/10.4121/uuid:82c1dc0d-5485-43d8-901a-ce7f26cda35d

Wind speeds are converted to 1-minute sustained winds through division by 0.88 (this value is taken from Bloemendaal et al. (2020), cited above).

Parameters

- **path** (*str*) – Full path to a txt-file as contained in the *data.zip* archive from the official source linked above.
- **years** (*list of int, optional*) – If given, only read the specified "years" from the txt-File. Note that a "year" refers to one ensemble of tracks in the data set that represents one sample year.

Returns
tracks – TCTracks with data from the STORM simulations.

Return type
TCTracks

Notes

All tracks are set in the year 1980. The id of the year (starting from 0) is saved in the attribute 'id_no'. To obtain the year of each track use

```
>>> years = [int(tr.attrs['id_no'] / 1000) for tr in tc_tracks.data]
>>> # or, alternatively,
>>> years = [int(tr.attrs['sid'].split("-")[-2]) for tr in tc_tracks.data]
```

If a windfield is generated from these tracks using the method `TropCylcone.from_tracks()`, the following should be considered:

1. The frequencies will be set to `1` for each storm. Thus, in order to compute annual values, the frequencies of the TropCylone should be changed to `1/number of years`.
2. The storm year and the storm id are stored in the `TropCyclone.event_name` attribute.

equal_timestep(*time_step_h=1*, *land_params=False*, *pool=None*)

Resample all tracks at the specified temporal resolution

The resulting track data will be given at evenly distributed time steps, relative to midnight (00:00). For example, if *time_step_h* is 1 and the original track data starts at 06:30, the interpolated track will not have a time step at 06:30 because only multiples of 01:00 (relative to midnight) are included. In this case, the interpolated track will start at 07:00.

Depending on the original resolution of the track data, this method may up- or downsample track time steps.

Note that tracks that already have the specified resolution remain unchanged.

Parameters

- **time_step_h** (*float or int, optional*) – Temporal resolution in hours (positive, may be non-integer-valued). Default: 1.
- **land_params** (*bool, optional*) – If True, recompute *on_land* and *dist_since_lf* at each node. Default: False.
- **pool** (*pathos.pool, optional*) – Pool that will be used for parallel computation when applicable. If not given, the pool attribute of *self* will be used. Default: None

calc_random_walk(***kwargs*)

Deprecated. Use *TCTracks.calc_perturbed_trajectories* instead.

calc_perturbed_trajectories(***kwargs*)

See function in *climada.hazard.tc_tracks_synth*.

property size

Get longitude from coord array.

get_bounds(*deg_buffer=0.1*)

Get bounds as (lon_min, lat_min, lon_max, lat_max) tuple.

Parameters
deg_buffer (*float*) – A buffer to add around the bounding box

Returns
bounds

Return type
tuple (lon_min, lat_min, lon_max, lat_max)

property bounds

Exact bounds of trackset as tuple, no buffer.

get_extent(*deg_buffer=0.1*)

Get extent as (lon_min, lon_max, lat_min, lat_max) tuple.

Parameters
deg_buffer (*float*) – A buffer to add around the bounding box

Returns
extent

Return type
tuple (lon_min, lon_max, lat_min, lat_max)

property extent

Exact extent of trackset as tuple, no buffer.

generate_centroids(*res_deg*, *buffer_deg*)

Generate gridded centroids within padded bounds of tracks

Parameters

- **res_deg** (*float*) – Resolution in degrees.
- **buffer_deg** (*float*) – Buffer around tracks in degrees.

Returns
centroids – Centroids instance.

Return type
Centroids

plot(*axis=None*, *figsize=(9, 13)*, *legend=True*, *adapt_fontsize=True*, ***kwargs*)

Track over earth. Historical events are blue, probabilistic black.

Parameters

- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*(float, float), optional*) – figure size for plt.subplots The default is (9, 13)
- **legend** (*bool, optional*) – whether to display a legend of Tropical Cyclone categories. Default: True.
- **kwargs** (*optional*) – arguments for LineCollection matplotlib, e.g. alpha=0.5
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.

Returns
axis

Return type
matplotlib.axes._subplots.AxesSubplot

write_netcdf(*folder_name*)

Write a netcdf file per track with track.sid name in given folder.

Parameters
folder_name (*str*) – Folder name where to write files.

read_netcdf(**args*, ***kwargs*)

This function is deprecated, use TCTracks.from_netcdf instead.

classmethod from_netcdf(*folder_name*)

Create new TCTracks object from NetCDF files contained in a given folder

> **Warning:** Do not use this classmethod for reading IBTrACS NetCDF files! If you need to manually download IBTrACS NetCDF files, place them in the `~/climada/data/system` folder and use the `TCTracks.from_ibtracks_netcdf` classmethod.

Parameters
folder_name (*str*) – Folder name from where to read files.

Returns
tracks – TCTracks with data from the given directory of NetCDF files.

Return type
TCTracks

write_hdf5(*file_name*, *complevel=5*)
Write TC tracks in NetCDF4-compliant HDF5 format.

Parameters

- **file_name** (*str or Path*) – Path to a new HDF5 file. If it exists already, the file is overwritten.
- **complevel** (*int*) – Specifies a compression level (0-9) for the zlib compression of the data. A value of 0 or None disables compression. Default: 5

classmethod from_hdf5(*file_name*)
Create new TCTracks object from a NetCDF4-compliant HDF5 file

Parameters
file_name (*str or Path*) – Path to a file that has been generated with *TCTracks.write_hdf*.

Returns
tracks – TCTracks with data from the given HDF5 file.

Return type
TCTracks

to_geodataframe(*as_points=False*, *split_lines_antimeridian=True*)
Transform this TCTracks instance into a GeoDataFrame.

Parameters

- **as_points** (*bool, optional*) – If False (default), one feature (row) per track with a LineString or MultiLineString as geometry (or Point geometry for tracks of length one) and all track attributes (sid, name, orig_event_flag, etc) as dataframe columns. If True, one feature (row) per track time step, with variable values per time step (radius_max_wind, max_sustained_wind, etc) as columns in addition to attributes.
- **split_lines_antimeridian** (*bool, optional*) – If True, tracks that cross the antimeridian are split into multiple Lines as a MultiLineString, with each Line on either side of the meridian. This ensures all Lines are within (-180, +180) degrees longitude. Note that lines might be split at more locations than strictly necessary, due to the underlying splitting algorithm (https://github.com/Toblerity/Shapely/issues/572).

Returns
gdf

Return type
GeoDataFrame

climada.hazard.tc_tracks.**set_category**(*max_sus_wind*, *wind_unit='kn'*, *saffir_scale=None*)
Add storm category according to Saffir-Simpson hurricane scale.

Parameters

- **max_sus_wind** (*np.array*) – Maximum sustained wind speed records for a single track.
- **wind_unit** (*str, optional*) – Units of wind speed. Default: 'kn'.
- **saffir_scale** (*list, optional*) – Saffir-Simpson scale in same units as wind (default scale valid for knots).

Returns

category –

Intensity of given track according to the Saffir-Simpson hurricane scale:

- -1 : tropical depression
- 0 : tropical storm
- 1 : Hurricane category 1
- 2 : Hurricane category 2
- 3 : Hurricane category 3
- 4 : Hurricane category 4
- 5 : Hurricane category 5

Return type
int

5.3.8 climada.hazard.tc_tracks_synth module

climada.hazard.tc_tracks_synth.**LANDFALL_DECAY_V = {-1: 0.00012859077693295416, 0: 0.0017226346292718126, 1: 0.002309772914350468, 2: 0.0025968221565522698, 3: 0.002626252944053856, 4: 0.002550639312763181, 5: 0.003788695795963695}**

Global landfall decay parameters for wind speed by TC category.

Keys are TC categories with -1='TD', 0='TS', 1='Cat 1', …, 5='Cat 5'.

It is v_rel as derived from:

```
>>> tracks = TCTracks.from_ibtracs_netcdf(year_range=(1980,2019), estimate_
↪missing=True)
>>> extent = tracks.get_extent()
>>> land_geom = climada.util.coordinates.get_land_geometry(
...     extent=extent, resolution=10
... )
>>> v_rel, p_rel = _calc_land_decay(tracks.data, land_geom, pool=tracks.pool)
```

climada.hazard.tc_tracks_synth.**LANDFALL_DECAY_P = {-1: (1.0088807492745373, 0.002117478217863062), 0: (1.0192813768091684, 0.003068578025845065), 1: (1.0362982218631644, 0.003620816186262243), 2: (1.0468630800617038, 0.004067381088015585), 3: (1.0639055205005432, 0.003708174876364079), 4: (1.0828373148889825, 0.003997492773076179), 5: (1.1088615145002092, 0.005224331234796362)}**

Global landfall decay parameters for pressure by TC category.

Keys are TC categories with -1='TD', 0='TS', 1='Cat 1', …, 5='Cat 5'.

It is p_rel as derived from:

```
>>> tracks = TCTracks.from_ibtracs_netcdf(year_range=(1980,2019), estimate_
↪missing=True)
>>> extent = tracks.get_extent()
>>> land_geom = climada.util.coordinates.get_land_geometry(
...     extent=extent, resolution=10
... )
>>> v_rel, p_rel = _calc_land_decay(tracks.data, land_geom, pool=tracks.pool)
```

climada.hazard.tc_tracks_synth.**calc_perturbed_trajectories**(*tracks*, *nb_synth_tracks=9*, *max_shift_ini=0.75*, *max_dspeed_rel=0.3*, *max_ddirection=0.008726646259971648*, *autocorr_dspeed=0.85*, *autocorr_ddirection=0.5*, *seed=54*, *decay=True*, *use_global_decay_params=True*, *pool=None*)

Generate synthetic tracks based on directed random walk. An ensemble of nb_synth_tracks synthetic tracks is computed for every track contained in self.

The methodology perturbs the tracks locations, and if decay is True it additionally includes decay of wind speed and central pressure drop after landfall. No other track parameter is perturbed. The track starting point location is perturbed by random uniform values of magnitude up to max_shift_ini in both longitude and latitude. Then, each segment between two consecutive points is perturbed in direction and distance (i.e., translational speed). These perturbations can be correlated in time, i.e., the perturbation in direction applied to segment i is correlated with the perturbation in direction applied to segment i-1 (and similarly for the perturbation in translational speed). Perturbations in track direction and temporal auto-correlations in perturbations are on an hourly basis, and the perturbations in translational speed is relative. Hence, the parameter values are relatively insensitive to the temporal resolution of the tracks. Note however that all tracks should be at the same temporal resolution, which can be achieved using equal_timestep(). max_dspeed_rel and autocorr_dspeed control the spread along the track ('what distance does the track run for'), while max_ddirection and autocorr_ddirection control the spread perpendicular to the track movement ('how does the track diverge in direction'). max_dspeed_rel and max_ddirection control the amplitude of perturbations at each track timestep but perturbations may tend to compensate each other over time, leading to a similar location at the end of the track, while autocorr_dspeed and autocorr_ddirection control how these perturbations persist in time and hence the amplitude of the perturbations towards the end of the track.

Note that the default parameter values have been only roughly calibrated so that the frequency of tracks in each 5x5degree box remains approximately constant. This is not an in-depth calibration and should be treated as such. The object is mutated in-place.

Parameters

- **tracks** (*climada.hazard.TCTracks*) – Tracks data.
- **nb_synth_tracks** (*int, optional*) – Number of ensemble members per track. Default: 9.
- **max_shift_ini** (*float, optional*) – Amplitude of max random starting point shift in decimal degree (up to +/-max_shift_ini for longitude and latitude). Default: 0.75.
- **max_dspeed_rel** (*float, optional*) – Amplitude of translation speed perturbation in relative terms (e.g., 0.2 for +/-20%). Default: 0.3.
- **max_ddirection** (*float, optional*) – Amplitude of track direction (bearing angle) perturbation per hour, in radians. Default: pi/360.
- **autocorr_dspeed** (*float, optional*) – Temporal autocorrelation in translation speed perturbation at a lag of 1 hour. Default: 0.85.
- **autocorr_ddirection** (*float, optional*) – Temporal autocorrelation of translational direction perturbation at a lag of 1 hour. Default: 0.5.
- **seed** (*int, optional*) – Random number generator seed for replicability of random walk. Put negative value if you don't want to use it. Default: configuration file.
- **decay** (*bool, optional*) – Whether to apply landfall decay in probabilistic tracks. Default: True.
- **use_global_decay_params** (*bool, optional*) – Whether to use precomputed global parameter values for landfall decay obtained from IBTrACS (1980-2019). If False, parameters are fitted

using historical tracks in input parameter 'tracks', in which case the landfall decay applied depends on the tracks passed as an input and may not be robust if few historical tracks make landfall in this object. Default: True.

- **pool** (*pathos.pool, optional*) – Pool that will be used for parallel computation when applicable. If not given, the pool attribute of *tracks* will be used. Default: None

5.3.9 climada.hazard.trop_cyclone module

class climada.hazard.trop_cyclone.**TropCyclone**(*category: ndarray | None = None*, *basin: List | None = None*, *windfields: List[csr_matrix] | None = None*, ***kwargs*)

Bases: `Hazard`

Contains tropical cyclone events.

category

for every event, the TC category using the Saffir-Simpson scale:

- -1 tropical depression
- 0 tropical storm
- 1 Hurrican category 1
- 2 Hurrican category 2
- 3 Hurrican category 3
- 4 Hurrican category 4
- 5 Hurrican category 5

Type

np.ndarray of ints

basin

Basin where every event starts:

- 'NA' North Atlantic
- 'EP' Eastern North Pacific
- 'WP' Western North Pacific
- 'NI' North Indian
- 'SI' South Indian
- 'SP' Southern Pacific
- 'SA' South Atlantic

Type

list of str

windfields

For each event, the full velocity vectors at each centroid and track position in a sparse matrix of shape (npositions, ncentroids * 2) that can be reshaped to a full ndarray of shape (npositions, ncentroids, 2).

Type
list of csr_matrix

intensity_thres = 17.5

intensity threshold for storage in m/s

vars_opt = {'category'}

Name of the variables that are not needed to compute the impact.

__init__(*category: ndarray | None = None*, *basin: List | None = None*, *windfields: List[csr_matrix] | None = None*, ***kwargs*)

Initialize values.

Parameters

- **category** (*np.ndarray of int, optional*) –

  For every event, the TC category using the Saffir-Simpson scale:
  -1 tropical depression 0 tropical storm 1 Hurrican category 1 2 Hurrican category 2 3 Hurrican category 3 4 Hurrican category 4 5 Hurrican category 5

- **basin** (*list of str, optional*) –

  Basin where every event starts:
  'NA' North Atlantic 'EP' Eastern North Pacific 'WP' Western North Pacific 'NI' North Indian 'SI' South Indian 'SP' Southern Pacific 'SA' South Atlantic

- **windfields** (*list of csr_matrix, optional*) – For each event, the full velocity vectors at each centroid and track position in a sparse matrix of shape (npositions, ncentroids * 2) that can be reshaped to a full ndarray of shape (npositions, ncentroids, 2).

- ****kwargs** (*Hazard properties, optional*) – All other keyword arguments are passed to the Hazard constructor.

set_from_tracks(**args*, ***kwargs*)

This function is deprecated, use TropCyclone.from_tracks instead.

classmethod from_tracks(*tracks: TCTracks*, *centroids: Centroids | None = None*, *pool: ProcessPool | None = None*, *model: str = 'H08'*, *ignore_distance_to_coast: bool = False*, *store_windfields: bool = False*, *metric: str = 'equirect'*, *intensity_thres: float = 17.5*, *max_latitude: float = 61*, *max_dist_inland_km: float = 1000*, *max_dist_eye_km: float = 300*, *max_memory_gb: float = 8*)

Create new TropCyclone instance that contains windfields from the specified tracks.

This function sets the `intensity` attribute to contain, for each centroid, the maximum wind speed (1-minute sustained winds at 10 meters above ground) experienced over the whole period of each TC event in m/s. The wind speed is set to 0 if it doesn't exceed the threshold `intensity_thres`.

The `category` attribute is set to the value of the `category`-attribute of each of the given track data sets.

The `basin` attribute is set to the genesis basin for each event, which is the first value of the `basin`-variable in each of the given track data sets.

Optionally, the time dependent, vectorial winds can be stored using the `store_windfields` function parameter (see below).

Parameters

- **tracks** (*climada.hazard.TCTracks*) – Tracks of storm events.
- **centroids** (*Centroids, optional*) – Centroids where to model TC. Default: global centroids at 360 arc-seconds resolution.

- **pool** (*pathos.pool, optional*) – Pool that will be used for parallel computation of wind fields. Default: None
- **description** (*str, optional*) – Description of the event set. Default: "".
- **model** (*str, optional*) – Parametric wind field model to use: one of "H1980" (the prominent Holland 1980 model), "H08" (Holland 1980 with b-value from Holland 2008), "H10" (Holland et al. 2010), or "ER11" (Emanuel and Rotunno 2011). Default: "H08".
- **ignore_distance_to_coast** (*boolean, optional*) – If True, centroids far from coast are not ignored. Default: False.
- **store_windfields** (*boolean, optional*) – If True, the Hazard object gets a list `windfields` of sparse matrices. For each track, the full velocity vectors at each centroid and track position are stored in a sparse matrix of shape (npositions, ncentroids * 2) that can be reshaped to a full ndarray of shape (npositions, ncentroids, 2). Default: False.
- **metric** (*str, optional*) – Specify an approximation method to use for earth distances:
  - "equirect": Distance according to sinusoidal projection. Fast, but inaccurate for large distances and high latitudes.
  - "geosphere": Exact spherical distance. Much more accurate at all distances, but slow.

  Default: "equirect".
- **intensity_thres** (*float, optional*) – Wind speeds (in m/s) below this threshold are stored as 0. Default: 17.5
- **max_latitude** (*float, optional*) – No wind speed calculation is done for centroids with latitude larger than this parameter. Default: 61
- **max_dist_inland_km** (*float, optional*) – No wind speed calculation is done for centroids with a distance (in km) to the coast larger than this parameter. Default: 1000
- **max_dist_eye_km** (*float, optional*) – No wind speed calculation is done for centroids with a distance (in km) to the TC center ("eye") larger than this parameter. Default: 300
- **max_memory_gb** (*float, optional*) – To avoid memory issues, the computation is done for chunks of the track sequentially. The chunk size is determined depending on the available memory (in GB). Note that this limit applies to each thread separately if a `pool` is used. Default: 8

Raises
`ValueError` –

Return type
TropCyclone

`apply_climate_scenario_knu`(*ref_year: int = 2050*, *rcp_scenario: int = 45*)

From current TC hazard instance, return new hazard set with future events for a given RCP scenario and year based on the parametrized values derived from Table 3 in Knutson et al 2015. https://doi.org/10.1175/JCLI-D-15-0129.1 . The scaling for different years and RCP scenarios is obtained by linear interpolation.

Note: The parametrized values are derived from the overall changes in statistical ensemble of tracks. Hence, this method should only be applied to sufficiently large tropical cyclone event sets that approximate the reference years 1981 - 2008 used in Knutson et. al.

The frequency and intensity changes are applied independently from one another. The mean intensity factors can thus slightly deviate from the Knutson value (deviation was found to be less than 1% for default IBTrACS event sets 1980-2020 for each basin).

Parameters

- **ref_year** (*int*) – year between 2000 ad 2100. Default: 2050
- **rcp_scenario** (*int*) – 26 for RCP 2.6, 45 for RCP 4.5, 60 for RCP 6.0 and 85 for RCP 8.5. The default is 45.

Returns
haz_cc – Tropical cyclone with frequencies and intensity scaled according to the Knutson criterion for the given year and RCP. Returns a new instance of climada.hazard.TropCyclone, self is not modified.

Return type
climada.hazard.TropCyclone

set_climate_scenario_knu(**args*, ***kwargs*)

This function is deprecated, use TropCyclone.apply_climate_scenario_knu instead.

classmethod video_intensity(*track_name: str*, *tracks: ~climada.hazard.tc_tracks.TCTracks*, *centroids: ~climada.hazard.centroids.centr.Centroids*, *file_name: str | None = None*, *writer: <module 'matplotlib.animation' from '/home/docs/checkouts/readthedocs.org/user_builds/climada-python/conda/stable/lib/python3.9/site-packages/matplotlib/animation.py'> = <matplotlib.animation.PillowWriter object>*, *figsize: ~typing.Tuple[float*, *float] = (9*, *13)*, *adapt_fontsize: bool = True*, ***kwargs*)

Generate video of TC wind fields node by node and returns its corresponding TropCyclone instances and track pieces.

Parameters

- **track_name** (*str*) – name of the track contained in tracks to record
- **tracks** (*climada.hazard.TCTracks*) – tropical cyclone tracks
- **centroids** (*climada.hazard.Centroids*) – centroids where wind fields are mapped
- **file_name** (*str, optional*) – file name to save video (including full path and file extension)
- **writer** (*matplotlib.animation.*, optional*) – video writer. Default is pillow with bitrate=500
- **figsize** (*tuple, optional*) – figure size for plt.subplots
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **kwargs** (*optional*) – arguments for pcolormesh matplotlib function used in event plots

Returns
tc_list, tc_coord

Return type
list(*TropCyclone*), list(np.ndarray)

Raises
ValueError –

frequency_from_tracks(*tracks: List*)

Set hazard frequency from tracks data.

Parameters
tracks (*list of xarray.Dataset*)

classmethod from_single_track(*track: Dataset*, *centroids:* Centroids, *idx_centr_filter: ndarray*, *model: str = 'H08'*, *store_windfields: bool = False*, *metric: str = 'equirect'*, *intensity_thres: float = 17.5*, *max_dist_eye_km: float = 300*, *max_memory_gb: float = 8*)

Generate windfield hazard from a single track dataset

Parameters

- **track** (*xr.Dataset*) – Single tropical cyclone track.
- **centroids** (*Centroids*) – Centroids instance.
- **idx_centr_filter** (*np.ndarray*) – Indices of centroids to restrict to (e.g. sufficiently close to coast).
- **model** (*str, optional*) – Parametric wind field model, one of "H1980" (the prominent Holland 1980 model), "H08" (Holland 1980 with b-value from Holland 2008), "H10" (Holland et al. 2010), or "ER11" (Emanuel and Rotunno 2011). Default: "H08".
- **store_windfields** (*boolean, optional*) – If True, store windfields. Default: False.
- **metric** (*str, optional*) – Specify an approximation method to use for earth distances: "equirect" (faster) or "geosphere" (more accurate). See `dist_approx` function in `climada.util.coordinates`. Default: "equirect".
- **intensity_thres** (*float, optional*) – Wind speeds (in m/s) below this threshold are stored as 0. Default: 17.5
- **max_dist_eye_km** (*float, optional*) – No wind speed calculation is done for centroids with a distance (in km) to the TC center ("eye") larger than this parameter. Default: 300
- **max_memory_gb** (*float, optional*) – To avoid memory issues, the computation is done for chunks of the track sequentially. The chunk size is determined depending on the available memory (in GB). Default: 8

Raises

ValueError, KeyError –

Returns

haz

Return type

TropCyclone

5.4 climada.util package

5.4.1 climada.util.api_client module

class climada.util.api_client.**Download**(**args*, ***kwargs*)

Bases: `Model`

Database entry keeping track of downloaded files from the CLIMADA data API

url = <CharField: Download.url>

path = <CharField: Download.path>

startdownload = <DateTimeField: Download.startdownload>

enddownload = <DateTimeField: Download.enddownload>

exception Failed

Bases: `Exception`

The download failed for some reason.

DoesNotExist

alias of `DownloadDoesNotExist`

id = <AutoField: Download.id>

class `climada.util.api_client.`**FileInfo**(*uuid: str*, *url: str*, *file_name: str*, *file_format: str*, *file_size: int*, *check_sum: str*)

Bases: `object`

file data from CLIMADA data API.

uuid: str

url: str

file_name: str

file_format: str

file_size: int

check_sum: str

__init__(*uuid: str*, *url: str*, *file_name: str*, *file_format: str*, *file_size: int*, *check_sum: str*) → None

class `climada.util.api_client.`**DataTypeInfo**(*data_type: str*, *data_type_group: str*, *status: str*, *description: str*, *properties: list*, *key_reference: list | None = None*, *version_notes: list | None = None*)

Bases: `object`

data type meta data from CLIMADA data API.

data_type: str

data_type_group: str

status: str

description: str

properties: list

key_reference: list = None

version_notes: list = None

__init__(*data_type: str*, *data_type_group: str*, *status: str*, *description: str*, *properties: list*, *key_reference: list | None = None*, *version_notes: list | None = None*) → None

class `climada.util.api_client.`**DataTypeShortInfo**(*data_type: str*, *data_type_group: str*)

Bases: `object`

data type name and group from CLIMADA data API.

data_type: str

data_type_group: str

__init__(*data_type: str*, *data_type_group: str*) → None

class climada.util.api_client.**DatasetInfo**(*uuid: str*, *data_type: DataTypeShortInfo*, *name: str*, *version: str*, *status: str*, *properties: dict*, *files: list*, *doi: str*, *description: str*, *license: str*, *activation_date: str*, *expiration_date: str*)

Bases: `object`

dataset data from CLIMADA data API.

uuid: str

data_type: *DataTypeShortInfo*

name: str

version: str

status: str

properties: dict

files: list

doi: str

description: str

license: str

activation_date: str

expiration_date: str

static from_json(*jsono*)

creates a DatasetInfo object from the json object returned by the CLIMADA data api server.

Parameters
jsono (*dict*)

Return type
DatasetInfo

__init__(*uuid: str*, *data_type: DataTypeShortInfo*, *name: str*, *version: str*, *status: str*, *properties: dict*, *files: list*, *doi: str*, *description: str*, *license: str*, *activation_date: str*, *expiration_date: str*) → None

climada.util.api_client.**checksize**(*local_path*, *fileinfo*)

Checks sanity of downloaded file simply by comparing actual and registered size.

Parameters

- **local_path** (*Path*) – the downloaded file
- **filinfo** (*FileInfo*) – file information from CLIMADA data API

Raises
Download.Failed – if the file is not what it's supposed to be

`climada.util.api_client.`**`checkhash`**(*local_path*, *fileinfo*)

Checks sanity of downloaded file by comparing actual and registered check sum.

Parameters

- **local_path** (*Path*) – the downloaded file
- **filinfo** (*FileInfo*) – file information from CLIMADA data API

Raises

`Download.Failed` – if the file is not what it's supposed to be

`class` `climada.util.api_client.`**`Cacher`**(*cache_enabled*)

Bases: `object`

Utility class handling cached results from http requests, to enable the API Client working in offline mode.

`__init__`(*cache_enabled*)

Constructor of Cacher.

Parameters

cache_enabled (*bool, None*) – Default: None, in this case the value is taken from CONFIG.data_api.cache_enabled.

`store`(*result*, **args*, ***kwargs*)

stores the result from a API call to a local file.

The name of the file is the md5 hash of a string created from the call's arguments, the content of the file is the call's result in json format.

Parameters

- **result** (*dict*) – will be written in json format to the cached result file
- ***args** (*list of str*)
- ****kwargs** (*list of dict of (str,str)*)

`fetch`(**args*, ***kwargs*)

reloads the result from a API call from a local file, created by the corresponding call of *self.store*.

If no call with exactly the same arguments has been made in the past, the result is None.

Parameters

- ***args** (*list of str*)
- ****kwargs** (*list of dict of (str,str)*)

Return type

dict or None

`class` `climada.util.api_client.`**`Client`**(*cache_enabled=None*)

Bases: `object`

Python wrapper around REST calls to the CLIMADA data API server.

`MAX_WAITING_PERIOD = 6`

`UNLIMITED = 100000`

`DOWNLOAD_TIMEOUT = 3600`

`QUERY_TIMEOUT = 300`

exception AmbiguousResult

Bases: `Exception`

Custom Exception for Non-Unique Query Result

exception NoResult

Bases: `Exception`

Custom Exception for No Query Result

exception NoConnection

Bases: `Exception`

To be raised if there is no internet connection and no cached result.

__init__(*cache_enabled=None*)

Constructor of Client.

Data API host and chunk_size (for download) are configurable values. Default values are 'climada.ethz.ch' and 8096 respectively.

Parameters

cache_enabled (*bool, optional*) – This flag controls whether the api calls of this client are going to be cached to the local file system (location defined by CONFIG.data_api.cache_dir). If set to true, the client can reload the results from the cache in case there is no internet connection and thus work in offline mode. Default: None, in this case the value is taken from CONFIG.data_api.cache_enabled.

list_dataset_infos(*data_type=None*, *name=None*, *version=None*, *properties=None*, *status='active'*)

Find all datasets matching the given parameters.

Parameters

- **data_type** (*str, optional*) – data_type of the dataset, e.g., 'litpop' or 'draught'
- **name** (*str, optional*) – the name of the dataset
- **version** (*str, optional*) – the version of the dataset, 'any' for all versions, 'newest' or None for the newest version meeting the requirements Default: None
- **properties** (*dict, optional*) – search parameters for dataset properties, by default None any property has a string for key and can be a string or a list of strings for value
- **status** (*str, optional*) – valid values are 'preliminary', 'active', 'expired', 'test_dataset' and None by default 'active'

Return type

list of *DatasetInfo*

get_dataset_info(*data_type=None*, *name=None*, *version=None*, *properties=None*, *status='active'*)

Find the one dataset that matches the given parameters.

Parameters

- **data_type** (*str, optional*) – data_type of the dataset, e.g., 'litpop' or 'draught'
- **name** (*str, optional*) – the name of the dataset
- **version** (*str, optional*) – the version of the dataset Default: newest version meeting the requirements
- **properties** (*dict, optional*) – search parameters for dataset properties, by default None any property has a string for key and can be a string or a list of strings for value

- **status** (*str, optional*) – valid values are 'preliminary', 'active', 'expired', 'test_dataset', None by default 'active'

Return type
DatasetInfo

Raises

- **AmbiguousResult** – when there is more than one dataset matching the search parameters
- **NoResult** – when there is no dataset matching the search parameters

get_dataset_info_by_uuid(*uuid*)
Returns the data from 'https://climada.ethz.ch/data-api/v1/dataset/{uuid}' as DatasetInfo object.

Parameters
uuid (*str*) – the universal unique identifier of the dataset

Return type
DatasetInfo

Raises
NoResult – if the uuid is not valid

list_data_type_infos(*data_type_group=None*)
Returns all data types from the climada data API belonging to a given data type group.

Parameters
data_type_group (*str, optional*) – name of the data type group, by default None

Return type
list of *DataTypeInfo*

get_data_type_info(*data_type*)
Returns the metadata of the data type with the given name from the climada data API.

Parameters
data_type (*str*) – data type name

Return type
DataTypeInfo

Raises
NoResult – if there is no such data type registered

download_dataset(*dataset*, *target_dir=PosixPath('/home/docs/climada/data')*, *organize_path=True*)
Download all files from a given dataset to a given directory.

Parameters

- **dataset** (*DatasetInfo*) – the dataset
- **target_dir** (*Path, optional*) – target directory for download, by default *climada.util.constants.SYSTEM_DIR*
- **organize_path** (*bool, optional*) – if set to True the files will end up in subdirectories of target_dir: [target_dir]/[data_type_group]/[data_type]/[name]/[version] by default True

Returns

- **download_dir** (*Path*) – the path to the directory containing the downloaded files, will be created if organize_path is True
- **downloaded_files** (*list of Path*) – the downloaded files themselves

Raises
`Exception` – when one of the files cannot be downloaded

`static purge_cache_db`(*local_path*)

Removes entry from the sqlite database that keeps track of files downloaded by *cached_download*. This may be necessary in case a previous attempt has failed in an uncontroled way (power outage or the like).

Parameters

- **local_path** (*Path*) – target destination
- **fileinfo** (*FileInfo*) – file object as retrieved from the data api

`get_hazard`(*hazard_type*, *name=None*, *version=None*, *properties=None*, *status='active'*, *dump_dir=PosixPath('/home/docs/climada/data')*)

Queries the data api for hazard datasets of the given type, downloads associated hdf5 files and turns them into a climada.hazard.Hazard object.

Parameters

- **hazard_type** (*str*) – Type of climada hazard.
- **name** (*str, optional*) – the name of the dataset
- **version** (*str, optional*) – the version of the dataset Default: newest version meeting the requirements
- **properties** (*dict, optional*) – search parameters for dataset properties, by default None any property has a string for key and can be a string or a list of strings for value
- **status** (*str, optional*) – valid values are 'preliminary', 'active', 'expired', 'test_dataset', None by default 'active'
- **dump_dir** (*str, optional*) – Directory where the files should be downoladed. Default: SYSTEM_DIR If the directory is the SYSTEM_DIR (as configured in climada.conf, i.g. ~/climada/data), the eventual target directory is organized into dump_dir > hazard_type > dataset name > version

Returns
The combined hazard object

Return type
climada.hazard.Hazard

`to_hazard`(*dataset*, *dump_dir=PosixPath('/home/docs/climada/data')*)

Downloads hdf5 files belonging to the given datasets reads them into Hazards and concatenates them into a single climada.Hazard object.

Parameters

- **dataset** (*DatasetInfo*) – Dataset to download and read into climada.Hazard object.
- **dump_dir** (*str, optional*) – Directory where the files should be downoladed. Default: SYSTEM_DIR (as configured in climada.conf, i.g. ~/climada/data). If the directory is the SYSTEM_DIR, the eventual target directory is organized into dump_dir > hazard_type > dataset name > version

Returns
The combined hazard object

Return type
climada.hazard.Hazard

get_exposures(*exposures_type*, *name=None*, *version=None*, *properties=None*, *status='active'*, *dump_dir=PosixPath('/home/docs/climada/data')*)

Queries the data api for exposures datasets of the given type, downloads associated hdf5 files and turns them into a climada.entity.exposures.Exposures object.

Parameters

- **exposures_type** (*str*) – Type of climada exposures.
- **name** (*str, optional*) – the name of the dataset
- **version** (*str, optional*) – the version of the dataset Default: newest version meeting the requirements
- **properties** (*dict, optional*) – search parameters for dataset properties, by default None any property has a string for key and can be a string or a list of strings for value
- **status** (*str, optional*) – valid values are 'preliminary', 'active', 'expired', 'test_dataset', None by default 'active'
- **dump_dir** (*str, optional*) – Directory where the files should be downoladed. Default: SYSTEM_DIR If the directory is the SYSTEM_DIR, the eventual target directory is organized into dump_dir > hazard_type > dataset name > version

Returns

The combined exposures object

Return type

climada.entity.exposures.Exposures

to_exposures(*dataset*, *dump_dir=PosixPath('/home/docs/climada/data')*)

Downloads hdf5 files belonging to the given datasets reads them into Exposures and concatenates them into a single climada.Exposures object.

Parameters

- **dataset** (*DatasetInfo*) – Dataset to download and read into climada.Exposures objects.
- **dump_dir** (*str, optional*) – Directory where the files should be downoladed. Default: SYSTEM_DIR (as configured in climada.conf, i.g. ~/climada/data). If the directory is the SYSTEM_DIR, the eventual target directory is organized into dump_dir > exposures_type > dataset name > version

Returns

The combined exposures object

Return type

climada.entity.exposures.Exposures

get_litpop(*country=None*, *exponents=(1, 1)*, *version=None*, *dump_dir=PosixPath('/home/docs/climada/data')*)

Get a LitPop `Exposures` instance on a 150arcsec grid with the default parameters: exponents = (1,1) and fin_mode = 'pc'.

Parameters

- **country** (*str, optional*) – Country name or iso3 codes for which to create the LitPop object. For creating a LitPop object over multiple countries, use `get_litpop` individually and concatenate using `LitPop.concat`, see Examples. If country is None a global LitPop instance is created. Defaut is None.

- **exponents** (*tuple of two integers, optional*) – Defining power with which lit (nightlights) and pop (gpw) go into LitPop. To get nightlights^3 without population count: (3, 0). To use population count alone: (0, 1). Default: (1, 1)
- **version** (*str, optional*) – the version of the dataset Default: newest version meeting the requirements
- **dump_dir** (*str*) – directory where the files should be downoladed. Default: SYSTEM_DIR

Returns
default litpop Exposures object

Return type
climada.entity.exposures.Exposures

Examples

Combined default LitPop object for Austria and Switzerland:

```
>>> client = Client()
>>> litpop_aut = client.get_litpop("AUT")
>>> litpop_che = client.get_litpop("CHE")
>>> litpop_comb = LitPop.concat([litpop_aut, litpop_che])
```

get_centroids (*res_arcsec_land=150, res_arcsec_ocean=1800, extent=(-180, 180, -60, 60), country=None, version=None, dump_dir=PosixPath('/home/docs/climada/data')*)

Get centroids from teh API

Parameters

- **res_land_arcsec** (*int*) – resolution for land centroids in arcsec. Default is 150
- **res_ocean_arcsec** (*int*) – resolution for ocean centroids in arcsec. Default is 1800
- **country** (*str*) – country name, numeric code or iso code based on pycountry. Default is None (global).
- **extent** (*tuple*) – Format (min_lon, max_lon, min_lat, max_lat) tuple. If min_lon > lon_max, the extend crosses the antimeridian and is [lon_max, 180] + [-180, lon_min] Borders are inclusive. Default is (-180, 180, -60, 60).
- **version** (*str, optional*) – the version of the dataset Default: newest version meeting the requirements
- **dump_dir** (*str*) – directory where the files should be downoladed. Default: SYSTEM_DIR

Returns
Centroids from the api

Return type
climada.hazard.centroids.Centroids

static get_property_values (*dataset_infos, known_property_values=None, exclude_properties=None*)

Returns a dictionnary of possible values for properties of a data type, optionally given known property values.

Parameters

- **dataset_infos** (*list of DataSetInfo*) – as returned by list_dataset_infos
- **known_properties_value** (*dict, optional*) – dict {'property':'value1, 'property2':'value2'}, to provide only a subset of property values that can be combined with the given properties.

- **exclude_properties** (*list of str, optional*) – properties in this list will be excluded from the resulting dictionary, e.g., because they are strictly metadata and don't provide any information essential to the dataset. Default: 'creation_date', 'climada_version'

Returns
of possibles property values

Return type
dict

static into_datasets_df(*dataset_infos*)

Convenience function providing a DataFrame of datasets with properties.

Parameters
dataset_infos (*list of DatasetInfo*) – as returned by list_dataset_infos

Returns
of datasets with properties as found in query by arguments

Return type
pandas.DataFrame

static into_files_df(*dataset_infos*)

Convenience function providing a DataFrame of files aligned with the input datasets.

Parameters
datasets (*list of DatasetInfo*) – as returned by list_dataset_infos

Returns
of the files' informations including dataset informations

Return type
pandas.DataFrame

purge_cache(*target_dir=PosixPath('/home/docs/climada/data')*, *keep_testfiles=True*)

Removes downloaded dataset files from the given directory if they have been downloaded with the API client, if they are beneath the given directory and if one of the following is the case: - there status is neither 'active' nor 'test_dataset' - their status is 'test_dataset' and keep_testfiles is set to False - their status is 'active' and they are outdated, i.e., there is a dataset with the same

data_type and name but a newer version.

Parameters

- **target_dir** (*Path or str, optional*) – files downloaded beneath this directory and empty subdirectories will be removed. default: SYSTEM_DIR
- **keep_testfiles** (*bool, optional*) – if set to True, files from datasets with status 'test_dataset' will not be removed. default: True

get_dataset_file(***kwargs*)

Convenience method. Combines `get_dataset` and `download_dataset`. Returns the path to a single file if the dataset has only one, otherwise throws an error.

Parameters
****kwargs** – arguments for get_dataset and download_dataset

Return type
Path

5.4.2 climada.util.checker module

`climada.util.checker.size(exp_len, var, var_name)`

Check if the length of a variable is the expected one.

Raises

ValueError –

`climada.util.checker.shape(exp_row, exp_col, var, var_name)`

Check if the length of a variable is the expected one.

Raises

ValueError –

`climada.util.checker.array_optional(exp_len, var, var_name)`

Check if array has right size. Warn if array empty. Call check_size.

Parameters

- **exp_len** (*str*) – expected array size
- **var** (*np.array*) – numpy array to check
- **var_name** (*str*) – name of the variable. Used in error/warning msg

Raises

ValueError –

`climada.util.checker.array_default(exp_len, var, var_name, def_val)`

Check array has right size. Set default value if empty. Call check_size.

Parameters

- **exp_len** (*str*) – expected array size
- **var** (*np.array*) – numpy array to check
- **var_name** (*str*) – name of the variable. Used in error/warning msg
- **def_val** (*np.array*) – nump array used as default value

Raises

ValueError –

Return type

Filled array

5.4.3 climada.util.config module

5.4.4 climada.util.constants module

`climada.util.constants.SYSTEM_DIR = PosixPath('/home/docs/climada/data')`

Folder containing the data used internally

`climada.util.constants.DEMO_DIR = PosixPath('/home/docs/climada/demo/data')`

Folder containing the data used for tutorials

`climada.util.constants.ENT_DEMO_TODAY = PosixPath('/home/docs/climada/demo/data/demo_today.xlsx')`

Entity demo present in xslx format.

`climada.util.constants.`**`ENT_DEMO_FUTURE = PosixPath('/home/docs/climada/demo/data/demo_future_TEST.xlsx')`**

Entity demo future in xslx format.

`climada.util.constants.`**`HAZ_DEMO_MAT = PosixPath('/home/docs/climada/demo/data/atl_prob_nonames.mat')`**

hurricanes from 1851 to 2011 over Florida with 100 centroids.

Type
Hazard demo from climada in MATLAB

`climada.util.constants.`**`HAZ_DEMO_FL = PosixPath('/home/docs/climada/demo/data/SC22000_VE__M1.grd.gz')`**

Raster file of flood over Venezuela. Model from GAR2015

`climada.util.constants.`**`ENT_TEMPLATE_XLS = PosixPath('/home/docs/climada/data/entity_template.xlsx')`**

Entity template in xls format.

`climada.util.constants.`**`HAZ_TEMPLATE_XLS = PosixPath('/home/docs/climada/data/hazard_template.xlsx')`**

Hazard template in xls format.

`climada.util.constants.`**`ONE_LAT_KM = 111.12`**

Mean one latitude (in degrees) to km

`climada.util.constants.`**`EARTH_RADIUS_KM = 6371`**

Earth radius in km

`climada.util.constants.`**`GLB_CENTROIDS_MAT = PosixPath('/home/docs/climada/data/GLB_NatID_grid_0360as_adv_2.mat')`**

Global centroids

`climada.util.constants.`**`GLB_CENTROIDS_NC = PosixPath('/home/docs/climada/data/NatID_grid_0150as.nc')`**

For backwards compatibility, it remains available under its old name.

`climada.util.constants.`**`ISIMIP_GPWV3_NATID_150AS = PosixPath('/home/docs/climada/data/NatID_grid_0150as.nc')`**

Compressed version of National Identifier Grid in 150 arc-seconds from ISIMIP project, based on GPWv3. Location in ISIMIP repository:

ISIMIP2a/InputData/landuse_humaninfluences/population/ID_GRID/Nat_id_grid_ISIMIP.nc

More references:

- https://www.isimip.org/gettingstarted/input-data-bias-correction/details/13/
- https://sedac.ciesin.columbia.edu/data/set/gpw-v3-national-identifier-grid

`climada.util.constants.`**`NATEARTH_CENTROIDS = {150: PosixPath('/home/docs/climada/data/NatEarth_Centroids_150as.hdf5'), 360: PosixPath('/home/docs/climada/data/NatEarth_Centroids_360as.hdf5')}`**

Global centroids at XXX arc-seconds resolution, including region ids from Natural Earth. The 360 AS file includes distance to coast from NASA.

`climada.util.constants.`**`RIVER_FLOOD_REGIONS_CSV = PosixPath('/home/docs/climada/data/NatRegIDs.csv')`**

Look-up table for river flood module

`climada.util.constants.`**`TC_ANDREW_FL = PosixPath('/home/docs/climada/demo/data/ibtracs_global_intp-None_1992230N11325.csv')`**

Tropical cyclone Andrew in Florida

`climada.util.constants.`**`HAZ_DEMO_H5 = PosixPath('/home/docs/climada/demo/data/tc_fl_1990_2004.h5')`**

IBTrACS from 1990 to 2004 over Florida with 2500 centroids.

Type

Hazard demo in hdf5 format

`climada.util.constants.`**`EXP_DEMO_H5 = PosixPath('/home/docs/climada/demo/data/exp_demo_today.h5')`**

Exposures over Florida

`climada.util.constants.`**`WS_DEMO_NC = [PosixPath('/home/docs/climada/demo/data/fp_lothar_crop-test.nc'), PosixPath('/home/docs/climada/demo/data/fp_xynthia_crop-test.nc')]`**

Winter storm in Europe files. These test files have been generated using the netCDF kitchen sink:

```
>>> ncks -d latitude,50.5,54.0 -d longitude,3.0,7.5 ./file_in.nc ./file_out.nc
```

`climada.util.constants.`**`TEST_UNC_OUTPUT_IMPACT = 'test_unc_output_impact'`**

Demo uncertainty impact output

`climada.util.constants.`**`TEST_UNC_OUTPUT_COSTBEN = 'test_unc_output_costben'`**

Demo uncertainty costben output

5.4.5 climada.util.coordinates module

`climada.util.coordinates.`**`NE_EPSG = 4326`**

Natural Earth CRS EPSG

`climada.util.coordinates.`**`NE_CRS = 'epsg:4326'`**

Natural Earth CRS

`climada.util.coordinates.`**`TMP_ELEVATION_FILE = PosixPath('/home/docs/climada/data/tmp_elevation.tif')`**

Path of elevation file written in set_elevation

`climada.util.coordinates.`**`DEM_NODATA = -9999`**

Value to use for no data values in DEM, i.e see points

`climada.util.coordinates.`**`MAX_DEM_TILES_DOWN = 300`**

Maximum DEM tiles to dowload

`climada.util.coordinates.`**`NEAREST_NEIGHBOR_THRESHOLD = 100`**

Distance threshold in km for coordinate assignment. Nearest neighbors with greater distances are not considered.

climada.util.coordinates.**latlon_to_geosph_vector**(*lat*, *lon*, *rad=False*, *basis=False*)

Convert lat/lon coodinates to radial vectors (on geosphere)

Parameters

- **lat, lon** (*ndarrays of floats, same shape*) – Latitudes and longitudes of points.
- **rad** (*bool, optional*) – If True, latitude and longitude are not given in degrees but in radians.
- **basis** (*bool, optional*) – If True, also return an orthonormal basis of the tangent space at the given points in lat-lon coordinate system. Default: False.

Returns

- **vn** (*ndarray of floats, shape (..., 3)*) – Same shape as lat/lon input with additional axis for components.
- **vbasis** (*ndarray of floats, shape (..., 2, 3)*) – Only present, if *basis* is True. Same shape as lat/lon input with additional axes for components of the two basis vectors.

climada.util.coordinates.**lon_normalize**(*lon*, *center=0.0*)

Normalizes degrees such that always -180 < lon - center <= 180

The input data is modified in place!

Parameters

- **lon** (*np.array*) – Longitudinal coordinates
- **center** (*float, optional*) – Central longitude value to use instead of 0. If None, the central longitude is determined automatically.

Returns

lon – Normalized longitudinal coordinates. Since the input *lon* is modified in place (!), the returned array is the same Python object (instead of a copy).

Return type

np.array

climada.util.coordinates.**lon_bounds**(*lon*, *buffer=0.0*)

Bounds of a set of degree values, respecting the periodicity in longitude

The longitudinal upper bound may be 180 or larger to make sure that the upper bound is always larger than the lower bound. The lower longitudinal bound will never lie below -180 and it will only assume the value -180 if the specified buffering enforces it.

Note that, as a consequence of this, the returned bounds do not satisfy the inequality *lon_min* <= *lon* <= *lon_max* in general!

Usually, an application of this function is followed by a renormalization of longitudinal values around the longitudinal middle value:

```
>>> bounds = lon_bounds(lon)
>>> lon_mid = 0.5 * (bounds[0] + bounds[2])
>>> lon = lon_normalize(lon, center=lon_mid)
>>> np.all((bounds[0] <= lon) & (lon <= bounds[2]))
```

If the bounds cover a full circle (360 degrees), this function will always return (-180, 180), instead of (0, 360) or similar equivalent outputs.

Example

```
>>> lon_bounds(np.array([-179, 175, 178]))
(175, 181)
>>> lon_bounds(np.array([-179, 175, 178]), buffer=1)
(174, 182)
```

Parameters

- **lon** (*np.array*) – Longitudinal coordinates
- **buffer** (*float, optional*) – Buffer to add to both sides of the bounding box. Default: 0.0.

Returns
bounds – Bounding box of the given points.

Return type
tuple (lon_min, lon_max)

climada.util.coordinates.**latlon_bounds**(*lat*, *lon*, *buffer=0.0*)

Bounds of a set of degree values, respecting the periodicity in longitude

See *lon_bounds* for more information about the handling of longitudinal values crossing the antimeridian.

Example

```
>>> latlon_bounds(np.array([0, -2, 5]), np.array([-179, 175, 178]))
(175, -2, 181, 5)
>>> latlon_bounds(np.array([0, -2, 5]), np.array([-179, 175, 178]), buffer=1)
(174, -3, 182, 6)
```

Parameters

- **lat** (*np.array*) – Latitudinal coordinates
- **lon** (*np.array*) – Longitudinal coordinates
- **buffer** (*float, optional*) – Buffer to add to all sides of the bounding box. Default: 0.0.

Returns
bounds – Bounding box of the given points.

Return type
tuple (lon_min, lat_min, lon_max, lat_max)

climada.util.coordinates.**toggle_extent_bounds**(*bounds_or_extent*)

Convert between the "bounds" and the "extent" description of a bounding box

The difference between the two conventions is in the order in which the bounds for each coordinate direction are given. To convert from one description to the other, the two central entries of the 4-tuple are swapped. Hence, the conversion is symmetric.

Parameters
bounds_or_extent (*tuple (a, b, c, d)*) – Bounding box of the given points in "bounds" (or "extent") convention.

Returns
extent_or_bounds – Bounding box of the given points in "extent" (or "bounds") convention.

Return type
tuple (a, c, b, d)

`climada.util.coordinates.`**`dist_approx`**(*lat1*, *lon1*, *lat2*, *lon2*, *log=False*, *normalize=True*, *method='equirect'*, *units='km'*)

Compute approximation of geodistance in specified units

Several batches of points can be processed at once for improved performance. The distances of all (lat1, lon1)-points within a batch to all (lat2, lon2)-points within the same batch are computed, according to the formula:

result[k, i, j] = dist((lat1[k, i], lon1[k, i]), (lat2[k, j], lon2[k, j]))

Hence, each of lat1, lon1, lat2, lon2 is expected to be a 2-dimensional array and the resulting array will always be 3-dimensional.

Parameters

- **lat1, lon1** (*ndarrays of floats, shape (nbatch, nx)*) – Latitudes and longitudes of first points.
- **lat2, lon2** (*ndarrays of floats, shape (nbatch, ny)*) – Latitudes and longitudes of second points.
- **log** (*bool, optional*) – If True, return the tangential vectors at the first points pointing to the second points (Riemannian logarithm). Default: False.
- **normalize** (*bool, optional*) – If False, assume that all longitudinal values lie within a single interval of size 360 (e.g., between -180 and 180, or between 0 and 360) and such that the shortest path between any two points does not cross the antimeridian according to that parametrization. If True, a suitable interval is determined using `lon_bounds()` and the longitudinal values are reparametrized accordingly using `lon_normalize()`. Note that this option has no effect when using the "geosphere" method because it is independent from the parametrization. Default: True
- **method** (*str, optional*) – Specify an approximation method to use:
  - "equirect": Distance according to sinusoidal projection. Fast, but inaccurate for large distances and high latitudes.
  - "geosphere": Exact spherical distance. Much more accurate at all distances, but slow.

  Note that ellipsoidal distances would be even more accurate, but are currently not implemented. Default: "equirect".
- **units** (*str, optional*) – Specify a unit for the distance. One of:
  - "km": distance in km.
  - "m": distance in m.
  - "degree": angular distance in decimal degrees.
  - "radian": angular distance in radians.

  Default: "km".

Returns

- **dists** (*ndarray of floats, shape (nbatch, nx, ny)*) – Approximate distances in specified units.
- **vtan** (*ndarray of floats, shape (nbatch, nx, ny, 2)*) – If *log* is True, tangential vectors at first points in local lat-lon coordinate system.

`climada.util.coordinates.`**`compute_geodesic_lengths`**(*gdf*)

Calculate the great circle (geodesic / spherical) lengths along any (complicated) line geometry object, based on the pyproj.Geod implementation.

Parameters
gdf (*gpd.GeoDataframe with geometrical shapes of which to compute the length*)

Returns
series – objects in metres.

Return type
a pandas series (column) with the great circle lengths of the

See also:

dist_approx()
distance between individual lat/lon-points

Note: This implementation relies on non-projected (i.e. geographic coordinate systems that span the entire globe) CRS only, which results in sea-level distances and hence a certain (minor) level of distortion; cf. https://gis.stackexchange.com/questions/176442/what-is-the-real-distance-between-positions

`climada.util.coordinates.`**`get_gridcellarea`**(*lat*, *resolution=0.5*, *unit='ha'*)

The area covered by a grid cell is calculated depending on the latitude

- 1 degree = ONE_LAT_KM (111.12km at the equator)
- longitudal distance in km = ONE_LAT_KM*resolution*cos(lat)
- latitudal distance in km = ONE_LAT_KM*resolution
- area = longitudal distance * latitudal distance

Parameters

- **lat** (*np.array*) – Latitude of the respective grid cell
- **resolution** (*int, optional*) – raster resolution in degree (default: 0.5 degree)
- **unit** (*string, optional*) – unit of the output area (default: ha, alternatives: m2, km2)

`climada.util.coordinates.`**`grid_is_regular`**(*coord*)

Return True if grid is regular. If True, returns height and width.

Parameters
coord (*np.array*) – Each row is a lat-lon-pair.

Returns

- **regular** (*bool*) – Whether the grid is regular. Only in this case, the following width and height are reliable.
- **height** (*int*) – Height of the supposed grid.
- **width** (*int*) – Width of the supposed grid.

`climada.util.coordinates.`**`get_coastlines`**(*bounds=None*, *resolution=110*)

Get Polygones of coast intersecting given bounds

Parameters

- **bounds** (*tuple*) – min_lon, min_lat, max_lon, max_lat in EPSG:4326
- **resolution** (*float, optional*) – 10, 50 or 110. Resolution in m. Default: 110m, i.e. 1:110.000.000

Returns
coastlines – Polygons of coast intersecting given bounds.

Return type
GeoDataFrame

`climada.util.coordinates.`**`convert_wgs_to_utm`**`(`*lon*, *lat*`)`

Get EPSG code of UTM projection for input point in EPSG 4326

Parameters

- **lon** (*float*) – longitude point in EPSG 4326
- **lat** (*float*) – latitude of point (lat, lon) in EPSG 4326

Returns
epsg_code – EPSG code of UTM projection.

Return type
int

`climada.util.coordinates.`**`utm_zones`**`(`*wgs_bounds*`)`

Get EPSG code and bounds of UTM zones covering specified region

Parameters
wgs_bounds (*tuple*) – lon_min, lat_min, lon_max, lat_max

Returns
zones – EPSG code and bounding box in WGS coordinates.

Return type
list of pairs (zone_epsg, zone_wgs_bounds)

`climada.util.coordinates.`**`dist_to_coast`**`(`*coord_lat*, *lon=None*, *signed=False*`)`

Compute (signed) distance to coast from input points in meters.

Parameters

- **coord_lat** (*GeoDataFrame or np.array or float*) –

  One of the following:

  – GeoDataFrame with geometry column in epsg:4326

  – np.array with two columns, first for latitude of each point and second with longitude in epsg:4326

  – np.array with one dimension containing latitudes in epsg:4326

  – float with a latitude value in epsg:4326

- **lon** (*np.array or float, optional*) –

  One of the following:

  – np.array with one dimension containing longitudes in epsg:4326

  – float with a longitude value in epsg:4326

- **signed** (*bool*) – If True, distance is signed with positive values off shore and negative values on land. Default: False

Returns
dist – (Signed) distance to coast in meters.

Return type
np.array

`climada.util.coordinates.`**`dist_to_coast_nasa`**(*lat*, *lon*, *highres=False*, *signed=False*)

Read interpolated (signed) distance to coast (in m) from NASA data

Note: The NASA raster file is 300 MB and will be downloaded on first run!

Parameters

- **lat** (*np.array*) – latitudes in epsg:4326
- **lon** (*np.array*) – longitudes in epsg:4326
- **highres** (*bool, optional*) – Use full resolution of NASA data (much slower). Default: False.
- **signed** (*bool*) – If True, distance is signed with positive values off shore and negative values on land. Default: False

Returns
dist – (Signed) distance to coast in meters.

Return type
np.array

`climada.util.coordinates.`**`get_land_geometry`**(*country_names=None*, *extent=None*, *resolution=10*)

Get union of the specified (or all) countries or the points inside the extent.

Parameters

- **country_names** (*list, optional*) – list with ISO3 names of countries, e.g ['ZWE', 'GBR', 'VNM', 'UZB']
- **extent** (*tuple, optional*) – (min_lon, max_lon, min_lat, max_lat)
- **resolution** (*float, optional*) – 10, 50 or 110. Resolution in m. Default: 10m, i.e. 1:10.000.000

Returns
geom – Polygonal shape of union.

Return type
shapely.geometry.multipolygon.MultiPolygon

`climada.util.coordinates.`**`coord_on_land`**(*lat*, *lon*, *land_geom=None*)

Check if points are on land.

Parameters

- **lat** (*np.array*) – latitude of points in epsg:4326
- **lon** (*np.array*) – longitude of points in epsg:4326
- **land_geom** (*shapely.geometry.multipolygon.MultiPolygon, optional*) – If given, use these as profiles of land. Otherwise, the global landmass is used.

Returns
on_land – Entries are True if corresponding coordinate is on land and False otherwise.

Return type
np.array(bool)

`climada.util.coordinates.`**`nat_earth_resolution`**(*resolution*)

Check if resolution is available in Natural Earth. Build string.

Parameters
resolution (*int*) – resolution in millions, 110 == 1:110.000.000.

Returns
res_name – Natural Earth name of resolution (e.g. '110m')

Return type
str

Raises
ValueError –

`climada.util.coordinates.`**`get_country_geometries`**`(`*country_names=None*, *extent=None*, *resolution=10*`)`

Natural Earth country boundaries within given extent

If no arguments are given, simply returns the whole natural earth dataset.

Take heed: we assume WGS84 as the CRS unless the Natural Earth download utility from cartopy starts including the projection information. (They are saving a whopping 147 bytes by omitting it.) Same goes for UTF.

If extent is provided, longitude values in 'geom' will all lie within 'extent' longitude range. Therefore setting extent to e.g. [160, 200, -20, 20] will provide longitude values between 160 and 200 degrees.

Parameters

- **country_names** (*list, optional*) – list with ISO 3166 alpha-3 codes of countries, e.g ['ZWE', 'GBR', 'VNM', 'UZB']
- **extent** (*tuple, optional*) – (min_lon, max_lon, min_lat, max_lat) Extent, assumed to be in the same CRS as the natural earth data.
- **resolution** (*float, optional*) – 10, 50 or 110. Resolution in m. Default: 10m

Returns
geom – Natural Earth multipolygons of the specified countries, resp. the countries that lie within the specified extent.

Return type
GeoDataFrame

`climada.util.coordinates.`**`get_region_gridpoints`**`(`*countries=None*, *regions=None*, *resolution=150*, *iso=True*, *rect=False*, *basemap='natearth'*`)`

Get coordinates of gridpoints in specified countries or regions

Parameters

- **countries** (*list, optional*) – ISO 3166-1 alpha-3 codes of countries, or internal numeric NatID if *iso* is set to False.
- **regions** (*list, optional*) – Region IDs.
- **resolution** (*float, optional*) – Resolution in arc-seconds, either 150 (default) or 360.
- **iso** (*bool, optional*) – If True, assume that countries are given by their ISO 3166-1 alpha-3 codes (instead of the internal NatID). Default: True.
- **rect** (*bool, optional*) – If True, a rectangular box around the specified countries/regions is selected. Default: False.
- **basemap** (*str, optional*) – Choose between different data sources. Currently available: "isimip" and "natearth". Default: "natearth".

Returns

- **lat** (*np.array*) – Latitude of points in epsg:4326.
- **lon** (*np.array*) – Longitude of points in epsg:4326.

`climada.util.coordinates.`**`assign_grid_points`**`(*args, **kwargs)`

This function has been renamed, use `match_grid_points` instead.

`climada.util.coordinates.`**`match_grid_points`**`(x, y, grid_width, grid_height, grid_transform)`

To each coordinate in *x* and *y*, assign the closest centroid in the given raster grid

Make sure that your grid specification is relative to the same coordinate reference system as the *x* and *y* coordinates. In case of lon/lat coordinates, make sure that the longitudinal values are within the same longitudinal range (such as [-180, 180]).

If your grid is given by bounds instead of a transform, the functions *rasterio.transform.from_bounds* and *pts_to_raster_meta* might be helpful.

Parameters

- **x, y** (*np.array*) – x- and y-coordinates of points to assign coordinates to.
- **grid_width** (*int*) – Width (number of columns) of the grid.
- **grid_height** (*int*) – Height (number of rows) of the grid.
- **grid_transform** (*affine.Affine*) – Affine transformation defining the grid raster.

Returns

assigned_idx – Index into the flattened *grid*. Note that the value *-1* is used to indicate that no matching coordinate has been found, even though *-1* is a valid index in NumPy!

Return type

np.array of size equal to the size of x and y

`climada.util.coordinates.`**`assign_coordinates`**`(*args, **kwargs)`

This function has been renamed, use `match_coordinates` instead.

`climada.util.coordinates.`**`match_coordinates`**`(coords, coords_to_assign, distance='euclidean', threshold=100, **kwargs)`

To each coordinate in *coords*, assign a matching coordinate in *coords_to_assign*

If there is no exact match for some entry, an attempt is made to assign the geographically nearest neighbor. If the distance to the nearest neighbor exceeds *threshold*, the index *-1* is assigned.

Currently, the nearest neighbor matching works with lat/lon coordinates only. However, you can disable nearest neighbor matching by setting *threshold* to 0, in which case only exactly matching coordinates are assigned to each other.

Make sure that all coordinates are according to the same coordinate reference system. In case of lat/lon coordinates, the "haversine" distance is able to correctly compute the distance across the antimeridian. However, when exact matches are enforced with *threshold=0*, lat/lon coordinates need to be given in the same longitudinal range (such as (-180, 180)).

Parameters

- **coords** (*np.array with two columns*) – Each row is a geographical coordinate pair. The result's size will match this array's number of rows.
- **coords_to_assign** (*np.array with two columns*) – Each row is a geographical coordinate pair. The result will be an index into the rows of this array. Make sure that these coordinates use the same coordinate reference system as *coords*.
- **distance** (*str, optional*) – Distance to use for non-exact matching. Possible values are "euclidean", "haversine" and "approx". Default: "euclidean"
- **threshold** (*float, optional*) – If the distance to the nearest neighbor exceeds *threshold*, the index *-1* is assigned. Set *threshold* to 0 to disable nearest neighbor matching. Default: 100 (km)

- **kwargs** (*dict, optional*) – Keyword arguments to be passed on to nearest-neighbor finding functions in case of non-exact matching with the specified *distance*.

Returns
: **assigned_idx** – Index into *coords_to_assign*. Note that the value *-1* is used to indicate that no matching coordinate has been found, even though *-1* is a valid index in NumPy!

Return type
: np.array of size equal to the number of rows in *coords*

Notes

By default, the 'euclidean' distance metric is used to find the nearest neighbors in case of non-exact matching. This method is fast for (quasi-)gridded data, but introduces innacuracy since distances in lat/lon coordinates are not equal to distances in meters on the Earth surface, in particular for higher latitude and distances larger than 100km. If more accuracy is needed, please use the 'haversine' distance metric. This however is slower for (quasi-)gridded data.

climada.util.coordinates.**match_centroids**(*coord_gdf*, *centroids*, *distance='euclidean'*, *threshold=100*)

Assign to each gdf coordinate point its closest centroids's coordinate. If disatances > threshold in points' distances, -1 is returned. If centroids are in a raster and coordinate point is outside of it `-1` is assigned

Parameters

- **coord_gdf** (*gpd.GeoDataFrame*) – GeoDataframe with defined latitude/longitude column and crs
- **centroids** (*Centroids*) – (Hazard) centroids to match (as raster or vector centroids).
- **distance** (*str, optional*) – Distance to use in case of vector centroids. Possible values are "euclidean", "haversine" and "approx". Default: "euclidean"
- **threshold** (*float, optional*) – If the distance (in km) to the nearest neighbor exceeds *threshold*, the index *-1* is assigned. Set *threshold* to 0, to disable nearest neighbor matching. Default: `NEAREST_NEIGHBOR_THRESHOLD` (100km)

See also:

climada.util.coordinates.match_grid_points
: method to associate centroids to coordinate points when centroids is a raster

climada.util.coordinates.match_coordinates
: method to associate centroids to coordinate points

Notes

The default order of use is:

1. if centroid raster is defined, assign the closest raster point to each gdf coordinate point. 2. if no raster, assign centroids to the nearest neighbor using euclidian metric

Both cases can introduce innacuracies for coordinates in lat/lon coordinates as distances in degrees differ from distances in meters on the Earth surface, in particular for higher latitude and distances larger than 100km. If more accuracy is needed, please use 'haversine' distance metric. This however is slower for (quasi-)gridded data, and works only for non-gridded data.

climada.util.coordinates.**region2isos**(*regions*)

Convert region names to ISO 3166 alpha-3 codes of countries

Parameters

regions (*str or list of str*) – Region name(s).

Returns

isos – Sorted list of iso codes of all countries in specified region(s).

Return type

list of str

climada.util.coordinates.**country_to_iso**(*countries*, *representation='alpha3'*, *fillvalue=None*)

Determine ISO 3166 representation of countries

Example

```
>>> country_to_iso(840)
'USA'
>>> country_to_iso("United States", representation="alpha2")
'US'
>>> country_to_iso(["United States of America", "SU"], "numeric")
[840, 810]
```

Some geopolitical areas that are not covered by ISO 3166 are added in the "user-assigned" range of ISO 3166-compliant values:

```
>>> country_to_iso(["XK", "Dhekelia"], "numeric")  # XK for Kosovo
[983, 907]
```

Parameters

- **countries** (*one of str, int, list of str, list of int*) – Country identifiers: name, official name, alpha-2, alpha-3 or numeric ISO codes. Numeric representations may be specified as str or int.
- **representation** (*str (one of "alpha3", "alpha2", "numeric", "name"), optional*) – All countries are converted to this representation according to ISO 3166. Default: "alpha3".
- **fillvalue** (*str or int or None, optional*) – The value to assign if a country is not recognized by the given identifier. By default, a LookupError is raised. Default: None

Returns

iso_list – ISO 3166 representation of countries. Will only return a list if the input is a list. Numeric representations are returned as integers.

Return type

one of str, int, list of str, list of int

climada.util.coordinates.**country_iso_alpha2numeric**(*iso_alpha*)

Deprecated: Use *country_to_iso* with *representation="numeric"* instead

climada.util.coordinates.**country_natid2iso**(*natids*, *representation='alpha3'*)

Convert internal NatIDs to ISO 3166-1 alpha-3 codes

Parameters

- **natids** (*int or list of int*) – NatIDs of countries (or single ID) as used in ISIMIP's version of the GPWv3 national identifier grid.
- **representation** (*str, one of "alpha3", "alpha2" or "numeric"*) – All countries are converted to this representation according to ISO 3166. Default: "alpha3".

Returns
iso_list – ISO 3166 representation of countries. Will only return a list if the input is a list. Numeric representations are returned as integers.

Return type
one of str, int, list of str, list of int

`climada.util.coordinates.`**`country_iso2natid`**`(`*isos*`)`

Convert ISO 3166-1 alpha-3 codes to internal NatIDs

Parameters
isos (*str or list of str*) – ISO codes of countries (or single code).

Returns
natids – Will only return a list if the input is a list.

Return type
int or list of int

`climada.util.coordinates.`**`natearth_country_to_int`**`(`*country*`)`

Integer representation (ISO 3166, if possible) of Natural Earth GeoPandas country row

Parameters
country (*GeoSeries*) – Row from Natural Earth GeoDataFrame.

Returns
iso_numeric – Integer representation of given country.

Return type
int

`climada.util.coordinates.`**`get_country_code`**`(`*lat*, *lon*, *gridded=False*`)`

Provide numeric (ISO 3166) code for every point.

Oceans get the value zero. Areas that are not in ISO 3166 are given values in the range above 900 according to NATEARTH_AREA_NONISO_NUMERIC.

Parameters

- **lat** (*np.array*) – latitude of points in epsg:4326
- **lon** (*np.array*) – longitude of points in epsg:4326
- **gridded** (*bool*) – If True, interpolate precomputed gridded data which is usually much faster. Default: False.

Returns
country_codes – Numeric code for each point.

Return type
np.array(int)

`climada.util.coordinates.`**`get_admin1_info`**`(`*country_names*`)`

Provide Natural Earth registry info and shape files for admin1 regions

Parameters
country_names (*list or str*) – string or list with strings, either ISO code or names of countries, e.g.:

`['ZWE', 'GBR', 'VNM', 'UZB', 'Kenya', '051']` For example, for Armenia, all of the following inputs work: `'Armenia', 'ARM', 'AM', '051', 51`

Returns

- **admin1_info** (*dict*) – Data according to records in Natural Earth database.
- **admin1_shapes** (*dict*) – Shape according to Natural Earth.

`climada.util.coordinates.`**`get_admin1_geometries`**`(`*countries*`)`

return geometries, names and codes of admin 1 regions in given countries in a GeoDataFrame. If no admin 1 regions are defined, all regions in countries are returned.

Parameters

countries (*list or str or int*) – string or list with strings, either ISO code or names of countries, e.g.: `['ZWE', 'GBR', 'VNM', 'UZB', 'Kenya', '051']` For example, for Armenia, all of the following inputs work: `'Armenia', 'ARM', 'AM', '051', 51`

Returns

gdf –

geopandas.GeoDataFrame instance with columns:

"admin1_name"
[str] name of admin 1 region

"iso_3166_2"
[str] iso code of admin 1 region

"geometry"
[Polygon or MultiPolygon] shape of admin 1 region as shapely geometry object

"iso_3n"
[str] numerical iso 3 code of country (admin 0)

"iso_3a"
[str] alphabetical iso 3 code of country (admin 0)

Return type

GeoDataFrame

`climada.util.coordinates.`**`get_resolution_1d`**`(`*coords*, *min_resol=1e-08*`)`

Compute resolution of scalar grid

Parameters

- **coords** (*np.array*) – scalar coordinates
- **min_resol** (*float, optional*) – minimum resolution to consider. Default: 1.0e-8.

Returns

res – Resolution of given grid.

Return type

float

`climada.util.coordinates.`**`get_resolution`**`(`**coords*, *min_resol=1e-08*`)`

Compute resolution of n-d grid points

Parameters

- **X, Y, …** (*np.array*) – Scalar coordinates in each axis
- **min_resol** (*float, optional*) – minimum resolution to consider. Default: 1.0e-8.

Returns
resolution – Resolution in each coordinate direction.

Return type
pair of floats

`climada.util.coordinates.`**`pts_to_raster_meta`**(*points_bounds*, *res*)

Transform vector data coordinates to raster.

If a raster of the given resolution doesn't exactly fit the given bounds, the raster might have slightly larger (but never smaller) bounds.

Parameters

- **points_bounds** (*tuple*) – points total bounds (xmin, ymin, xmax, ymax)
- **res** (*tuple*) – resolution of output raster (xres, yres)

Returns

- **nrows** (*int*) – Number of rows.
- **ncols** (*int*) – Number of columns.
- **ras_trans** (*affine.Affine*) – Affine transformation defining the raster.

`climada.util.coordinates.`**`raster_to_meshgrid`**(*transform*, *width*, *height*)

Get coordinates of grid points in raster

Parameters

- **transform** (*affine.Affine*) – Affine transform defining the raster.
- **width** (*int*) – Number of points in first coordinate axis.
- **height** (*int*) – Number of points in second coordinate axis.

Returns

- **x** (*np.array of shape (height, width)*) – x-coordinates of grid points.
- **y** (*np.array of shape (height, width)*) – y-coordinates of grid points.

`climada.util.coordinates.`**`to_crs_user_input`**(*crs_obj*)

Returns a crs string or dictionary from a hdf5 file object.

bytes are decoded to str if the string starts with a '{' it is assumed to be a dumped string from a dictionary and ast is used to parse it.

Parameters
crs_obj (*int, dict or str or bytes*) – the crs object to be converted user input

Returns
to eventually be used as argument of rasterio.crs.CRS.from_user_input and pyproj.crs.CRS.from_user_input

Return type
str or dict

Raises
`ValueError` – if type(crs_obj) has the wrong type

`climada.util.coordinates.`**`equal_crs`**(*crs_one*, *crs_two*)

Compare two crs

Parameters

- **crs_one** (*dict, str or int*) – user crs
- **crs_two** (*dict, str or int*) – user crs

Returns
equal – Whether the two specified CRS are equal according tho rasterio.crs.CRS.from_user_input

Return type
bool

`climada.util.coordinates.`**`read_raster`**(*file_name*, *band=None*, *src_crs=None*, *window=None*, *geometry=None*, *dst_crs=None*, *transform=None*, *width=None*, *height=None*, *resampling='nearest'*)

Read raster of bands and set 0-values to the masked ones.

Parameters

- **file_name** (*str*) – name of the file
- **band** (*list(int), optional*) – band number to read. Default: 1
- **window** (*rasterio.windows.Window, optional*) – window to read
- **geometry** (*list of shapely.geometry, optional*) – consider pixels only within these shapes
- **dst_crs** (*crs, optional*) – reproject to given crs
- **transform** (*rasterio.Affine*) – affine transformation to apply
- **wdith** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampling** (*int or str, optional*) – Resampling method to use, encoded as an integer value (see *rasterio.enums.Resampling*). String values like *"nearest"* or *"bilinear"* are resolved to attributes of *rasterio.enums.Resampling*. Default: "nearest"

Returns

- **meta** (*dict*) – Raster meta (height, width, transform, crs).
- **data** (*np.array*) – Each row corresponds to one band (raster points are flattened, can be reshaped to height x width).

`climada.util.coordinates.`**`read_raster_bounds`**(*path*, *bounds*, *res=None*, *bands=None*, *resampling='nearest'*, *global_origin=None*, *pad_cells=1.0*)

Read raster file within given bounds at given resolution

By default, not only the grid cells of the destination raster whose cell centers fall within the specified bounds are selected, but one additional row/column of grid cells is added as a padding in each direction (pad_cells=1). This makes sure that the extent of the selected cell centers encloses the specified bounds.

The axis orientations (e.g. north to south, west to east) of the input data set are preserved.

Parameters

- **path** (*str*) – Path to raster file to open with rasterio.
- **bounds** (*tuple*) – (xmin, ymin, xmax, ymax)
- **res** (*float or pair of floats, optional*) – Resolution of output. Note that the orientation (sign) of these is overwritten by the input data set's axis orientations (e.g. north to south, west to east). Default: Resolution of input raster file.
- **bands** (*list of int, optional*) – Bands to read from the input raster file. Default: [1]

- **resampling** (*int or str, optional*) – Resampling method to use, encoded as an integer value (see *rasterio.enums.Resampling*). String values like *"nearest"* or *"bilinear"* are resolved to attributes of *rasterio.enums.Resampling*. Default: "nearest"
- **global_origin** (*pair of floats, optional*) – If given, align the output raster to a global reference raster with this origin. By default, the data set's origin (according to it's transform) is used.
- **pad_cells** (*float, optional*) – The number of cells to add as a padding (in terms of the destination grid that is inferred from *res* and/or *global_origin* if those parameters are given). This defaults to 1 to make sure that applying methods like bilinear interpolation to the output of this function is well-defined everywhere within the specified bounds. Default: 1.0

Returns

- **data** (*3d np.array*) – First dimension is for the selected raster bands. Second dimension is y (lat) and third dimension is x (lon).
- **transform** (*rasterio.Affine*) – Affine transformation defining the output raster data.

`climada.util.coordinates.`**`read_raster_sample`**(*path*, *lat*, *lon*, *intermediate_res=None*, *method='linear'*, *fill_value=None*)

Read point samples from raster file.

Parameters

- **path** (*str*) – path of the raster file
- **lat** (*np.array of shape (npoints,)*) – latitudes in file's CRS
- **lon** (*np.array of shape (npoints,)*) – longitudes in file's CRS
- **intermediate_res** (*float or pair of floats, optional*) – If given, the raster is not read in its original resolution but in the given one. This can increase performance for files of very high resolution.
- **method** (*str or pair of str, optional*) – The interpolation method, passed to *scipy.interpolate.interpn*. Default: 'linear'
- **fill_value** (*numeric, optional*) – The value used outside of the raster bounds. Default: The raster's nodata value or 0.

Returns

values – Interpolated raster values for each given coordinate point.

Return type

np.array of shape (npoints,)

`climada.util.coordinates.`**`read_raster_sample_with_gradients`**(*path*, *lat*, *lon*, *intermediate_res=None*, *method=('linear', 'nearest')*, *fill_value=None*)

Read point samples with computed gradients from raster file.

For convenience, and because this is the most common use case, the step sizes in the gradient computation are converted to meters if the raster's CRS is EPSG:4326 (lat/lon).

For example, in case of an elevation data set, not only the heights, but also the slopes of the terrain in x- and y-direction are returned. In addition, if the CRS of the elevation data set is EPSG:4326 (lat/lon) and elevations are given in m, then distances are converted from degrees to meters, so that the unit of the returned slopes is "meters (height) per meter (distance)".

Parameters

- **path** (*str*) – path of the raster file
- **lat** (*np.array of shape (npoints,)*) – latitudes in file's CRS
- **lon** (*np.array of shape (npoints,)*) – longitudes in file's CRS
- **intermediate_res** (*float or pair of floats, optional*) – If given, the raster is not read in its original resolution but in the given one. This can increase performance for files of very high resolution.
- **method** (*str or pair of str, optional*) – The interpolation methods for the data and its gradient, passed to *scipy.interpolate.interpn*. If a single string is given, the same interpolation method is used for both the data and its gradient. Default: ('linear', 'nearest')
- **fill_value** (*numeric, optional*) – The value used outside of the raster bounds. Default: The raster's nodata value or 0.

Returns

- **values** (*np.array of shape (npoints,)*) – Interpolated raster values for each given coordinate point.
- **gradient** (*np.array of shape (npoints, 2)*) – The raster gradient at each of the given coordinate points. The first/second value in each row is the derivative in lat/lon direction (lat is first!).

`climada.util.coordinates.`**`interp_raster_data`**(*data*, *interp_y*, *interp_x*, *transform*, *method='linear'*, *fill_value=0*)

Interpolate raster data, given as array and affine transform

Parameters

- **data** (*np.array*) – Array containing the values. The first two dimensions are always interpreted as corresponding to the y- and x-coordinates of the grid. Additional dimensions can be present in case of multi-band data.
- **interp_y** (*np.array*) – y-coordinates of points (corresp. to first axis of data)
- **interp_x** (*np.array*) – x-coordinates of points (corresp. to second axis of data)
- **transform** (*affine.Affine*) – affine transform defining the raster
- **method** (*str, optional*) – The interpolation method, passed to scipy.interpolate.interpn. Default: 'linear'.
- **fill_value** (*numeric, optional*) –

  The value used outside of the raster
  bounds. Default: 0.

Returns
values – Interpolated raster values for each given coordinate point. If multi-band data is provided, the additional dimensions from *data* will also be present in this array.

Return type
np.array

`climada.util.coordinates.`**`refine_raster_data`**(*data*, *transform*, *res*, *method='linear'*, *fill_value=0*)

Refine raster data, given as array and affine transform

Parameters

- **data** (*np.array*) – 2d array containing the values
- **transform** (*affine.Affine*) – affine transform defining the raster

- **res** (*float or pair of floats*) – new resolution
- **method** (*str, optional*) –

  The interpolation method, passed to
  scipy.interp.interpn. Default: 'linear'.

Returns

- **new_data** (*np.array*) – 2d array containing the interpolated values.
- **new_transform** (*affine.Affine*) – Affine transform defining the refined raster.

climada.util.coordinates.**read_vector**(*file_name*, *field_name*, *dst_crs=None*)

Read vector file format supported by fiona.

Parameters

- **file_name** (*str*) – vector file with format supported by fiona and 'geometry' field.
- **field_name** (*list(str)*) – list of names of the columns with values.
- **dst_crs** (*crs, optional*) – reproject to given crs

Returns

- **lat** (*np.array*) – Latitudinal coordinates.
- **lon** (*np.array*) – Longitudinal coordinates.
- **geometry** (*GeoSeries*) – Shape geometries.
- **value** (*np.array*) – Values associated to each shape.

climada.util.coordinates.**write_raster**(*file_name*, *data_matrix*, *meta*, *dtype=<class 'numpy.float32'>*)

Write raster in GeoTiff format.

Parameters

- **file_name** (*str*) – File name to write.
- **data_matrix** (*np.array*) – 2d raster data. Either containing one band, or every row is a band and the column represents the grid in 1d.
- **meta** (*dict*) – rasterio meta dictionary containing raster properties: width, height, crs and transform must be present at least. Include *compress="deflate"* for compressed output.
- **dtype** (*numpy dtype, optional*) – A numpy dtype. Default: np.float32

climada.util.coordinates.**points_to_raster**(*points_df*, *val_names=None*, *res=0.0*, *raster_res=0.0*, *crs='EPSG:4326'*, *scheduler=None*)

Compute raster (as data and transform) from GeoDataFrame.

Parameters

- **points_df** (*GeoDataFrame*) – contains columns latitude, longitude and those listed in the parameter *val_names*.
- **val_names** (*list of str, optional*) – The names of columns in *points_df* containing values. The raster will contain one band per column. Default: ['value']
- **res** (*float, optional*) – resolution of current data in units of latitude and longitude, approximated if not provided.
- **raster_res** (*float, optional*) – desired resolution of the raster

- **crs** (*object (anything accepted by pyproj.CRS.from_user_input), optional*) – If given, overwrites the CRS information given in *points_df*. If no CRS is explicitly given and there is no CRS information in *points_df*, the CRS is assumed to be EPSG:4326 (lat/lon). Default: None
- **scheduler** (*str*) – used for dask map_partitions. "threads", "synchronous" or "processes"

Returns

- **data** (*np.array*) – 3d array containing the raster values. The first dimension has the same size as *val_names* and represents the raster bands.
- **meta** (*dict*) – Dictionary with 'crs', 'height', 'width' and 'transform' attributes.

`climada.util.coordinates.`**`subraster_from_bounds`**(*transform*, *bounds*)

Compute a subraster definition from a given reference transform and bounds.

The axis orientations (sign of resolution step sizes) in *transform* are not required to be north to south and west to east. The given orientation is preserved in the result.

Parameters

- **transform** (*rasterio.Affine*) – Affine transformation defining the reference grid.
- **bounds** (*tuple of floats (xmin, ymin, xmax, ymax)*) – Bounds of the subraster in units and CRS of the reference grid.

Returns

- **dst_transform** (*rasterio.Affine*) – Subraster affine transformation. The axis orientations of the input transform (e.g. north to south, west to east) are preserved.
- **dst_shape** (*tuple of ints (height, width)*) – Number of pixels of subraster in vertical and horizontal direction.

`climada.util.coordinates.`**`align_raster_data`**(*source*, *src_crs*, *src_transform*, *dst_crs=None*, *dst_resolution=None*, *dst_bounds=None*, *global_origin=(-180, 90)*, *resampling='nearest'*, *conserve=None*, ***kwargs*)

Reproject 2D np.ndarray to be aligned to a reference grid.

This function ensures that reprojected data with the same dst_resolution and global_origins are aligned to the same global grid, i.e., no offset between destination grid points for different source grids that are projected to the same target resolution.

Note that the origin is required to be in the upper left corner. The result is always oriented left to right (west to east) and top to bottom (north to south).

Parameters

- **source** (*np.ndarray*) – The source is a 2D ndarray containing the values to be reprojected.
- **src_crs** (*CRS or dict*) – Source coordinate reference system, in rasterio dict format.
- **src_transform** (*rasterio.Affine*) – Source affine transformation.
- **dst_crs** (*CRS, optional*) – Target coordinate reference system, in rasterio dict format. Default: *src_crs*
- **dst_resolution** (*tuple (x_resolution, y_resolution) or float, optional*) – Target resolution (positive pixel sizes) in units of the target CRS. Default: *(abs(src_transform[0]), abs(src_transform[4]))*
- **dst_bounds** (*tuple of floats (xmin, ymin, xmax, ymax), optional*) – Bounds of the target raster in units of the target CRS. By default, the source's bounds are reprojected to the target CRS.

- **global_origin** (*tuple (west, north) of floats, optional*) – Coordinates of the reference grid's upper left corner. Default: (-180, 90). Make sure to change *global_origin* for non-geographical CRS!
- **resampling** (*int or str, optional*) – Resampling method to use, encoded as an integer value (see *rasterio.enums.Resampling*). String values like *"nearest"* or *"bilinear"* are resolved to attributes of *rasterio.enums.Resampling*. Default: "nearest"
- **conserve** (*str, optional*) – If provided, conserve the source array's 'mean' or 'sum' in the transformed data or normalize the values of the transformed data ndarray ('norm'). WARNING: Please note that this procedure will not apply any weighting of values according to the geographical cell sizes, which will introduce serious biases for lat/lon grids in case of areas spanning large latitudinal ranges. Default: None (no conservation)
- **kwargs** (*dict, optional*) – Additional arguments passed to *rasterio.warp.reproject*.

Raises
`ValueError` –

Returns

- **destination** (np.ndarray with same dtype as *source*) – The transformed 2D ndarray.
- **dst_transform** (*rasterio.Affine*) – Destination affine transformation.

`climada.util.coordinates.`**`mask_raster_with_geometry`**(*raster*, *transform*, *shapes*, *nodata=None*, ***kwargs*)

Change values in *raster* that are outside of given *shapes* to *nodata*.

This function is a wrapper for rasterio.mask.mask to allow for in-memory processing. This is done by first writing data to memfile and then reading from it before the function call to rasterio.mask.mask(). The MemoryFile will be discarded after exiting the with statement.

Parameters

- **raster** (*numpy.ndarray*) – raster to be masked with dim: [H, W].
- **transform** (*affine.Affine*) – the transform of the raster.
- **shapes** (*GeoJSON-like dict or an object that implements the Python geo*) – interface protocol (such as a Shapely Polygon) Passed to rasterio.mask.mask
- **nodata** (*int or float, optional*) – Passed to rasterio.mask.mask: Data points outside *shapes* are set to *nodata*.
- **kwargs** (*optional*) – Passed to rasterio.mask.mask.

Returns
masked – raster with dim: [H, W] and points outside shapes set to *nodata*

Return type
numpy.ndarray or numpy.ma.MaskedArray

`climada.util.coordinates.`**`set_df_geometry_points`**(*df_val*, *scheduler=None*, *crs=None*)

Set given geometry to given dataframe using dask if scheduler.

Parameters

- **df_val** (*GeoDataFrame*) – contains latitude and longitude columns
- **scheduler** (*str, optional*) – used for dask map_partitions. "threads", "synchronous" or "processes"

- **crs** (*object (anything readable by pyproj4.CRS.from_user_input), optional*) – Coordinate Reference System, if omitted or None: df_val.geometry.crs

`climada.util.coordinates.`**`fao_code_def`**`()`

Generates list of FAO country codes and corresponding ISO numeric-3 codes.

Returns

- **iso_list** (*list*) – list of ISO numeric-3 codes
- **faocode_list** (*list*) – list of FAO country codes

`climada.util.coordinates.`**`country_faocode2iso`**`(`*input_fao*`)`

Convert FAO country code to ISO numeric-3 codes.

Parameters
input_fao (*int or array*) – FAO country codes of countries (or single code)

Returns
output_iso – ISO numeric-3 codes of countries (or single code)

Return type
int or array

`climada.util.coordinates.`**`country_iso2faocode`**`(`*input_iso*`)`

Convert ISO numeric-3 codes to FAO country code.

Parameters
input_iso (*iterable of int*) – ISO numeric-3 code(s) of country/countries

Returns
output_faocode – FAO country code(s) of country/countries

Return type
numpy.array

5.4.6 climada.util.dates_times module

`climada.util.dates_times.`**`date_to_str`**`(`*date*`)`

Compute date string in ISO format from input datetime ordinal int. :Parameters: **date** (*int or list or np.array*) – input datetime ordinal

Return type
str or list(str)

`climada.util.dates_times.`**`str_to_date`**`(`*date*`)`

Compute datetime ordinal int from input date string in ISO format. :Parameters: **date** (*str or list*) – idate string in ISO format, e.g. '2018-04-06'

Return type
int

`climada.util.dates_times.`**`datetime64_to_ordinal`**`(`*datetime*`)`

Converts from a numpy datetime64 object to an ordinal date. See https://stackoverflow.com/a/21916253 for the horrible details. :Parameters: **datetime** (*np.datetime64, or list or np.array*) – date and time

Return type
int

`climada.util.dates_times.`**`last_year`**(*ordinal_vector*)

Extract first year from ordinal date

Parameters
ordinal_vector (*list or np.array*) – input datetime ordinal

Return type
int

`climada.util.dates_times.`**`first_year`**(*ordinal_vector*)

Extract first year from ordinal date

Parameters
ordinal_vector (*list or np.array*) – input datetime ordinal

Return type
int

5.4.7 climada.util.dwd_icon_loader module

`climada.util.dwd_icon_loader.`**`download_icon_grib`**(*run_datetime*, *model_name='icon-eu-eps'*, *parameter_name='vmax_10m'*, *max_lead_time=None*, *download_dir=None*)

download the gribfiles of a weather forecast run for a certain weather parameter from opendata.dwd.de/weather/nwp/.

Parameters

- **run_datetime** (*datetime*) – The starting timepoint of the forecast run
- **model_name** (*str*) – the name of the forecast model written as it appears in the folder structure in opendata.dwd.de/weather/nwp/ or 'test'
- **parameter_name** (*str*) – the name of the meteorological parameter written as it appears in the folder structure in opendata.dwd.de/weather/nwp/
- **max_lead_time** (*int*) – number of hours for which files should be downloaded, will default to maximum available data
- **download_dir** (*: str or Path*) – directory where the downloaded files should be saved in

Returns
file_names – a list of filenames that link to all just downloaded or available files from the forecast run, defined by the input parameters

Return type
list

`climada.util.dwd_icon_loader.`**`delete_icon_grib`**(*run_datetime*, *model_name='icon-eu-eps'*, *parameter_name='vmax_10m'*, *max_lead_time=None*, *download_dir=None*)

delete the downloaded gribfiles of a weather forecast run for a certain weather parameter from opendata.dwd.de/weather/nwp/.

Parameters

- **run_datetime** (*datetime*) – The starting timepoint of the forecast run
- **model_name** (*str*) – the name of the forecast model written as it appears in the folder structure in opendata.dwd.de/weather/nwp/

- **parameter_name** (*str*) – the name of the meteorological parameter written as it appears in the folder structure in opendata.dwd.de/weather/nwp/
- **max_lead_time** (*int*) – number of hours for which files should be deleted, will default to maximum available data
- **download_dir** (*str or Path*) – directory where the downloaded files are stored at the moment

`climada.util.dwd_icon_loader.`**`download_icon_centroids_file`**(*model_name='icon-eu-eps'*, *download_dir=None*)

create centroids based on netcdf files provided by dwd, links found here: https://www.dwd.de/DE/leistungen/opendata/neuigkeiten/opendata_dez2018_02.html https://www.dwd.de/DE/leistungen/opendata/neuigkeiten/opendata_aug2020_01.html

Parameters

- **model_name** (*str*) – the name of the forecast model written as it appears in the folder structure in opendata.dwd.de/weather/nwp/
- **download_dir** (*str or Path*) – directory where the downloaded files should be saved in

Returns

file_name – absolute path and filename of the downloaded and decompressed netcdf file

Return type

str

5.4.8 climada.util.earth_engine module

5.4.9 climada.util.files_handler module

`climada.util.files_handler.`**`to_list`**(*num_exp*, *values*, *val_name*)

Check size and transform to list if necessary. If size is one, build a list with num_exp repeated values.

Parameters

- **num_exp** (*int*) – expected number of list elements
- **values** (*object or list(object)*) – values to check and transform
- **val_name** (*str*) – name of the variable values

Return type

list

`climada.util.files_handler.`**`get_file_names`**(*file_name*)

Return list of files contained. Supports globbing.

Parameters

file_name (*str or list(str)*) – Either a single string or a list of strings that are either - a file path - or the path of the folder containing the files - or a globbing pattern.

Return type

list(str)

5.4.10 climada.util.finance module

`climada.util.finance.`**`net_present_value`**(*years*, *disc_rates*, *val_years*)

Compute net present value.

Parameters

- **years** (*np.array*) – array with the sequence of years to consider.
- **disc_rates** (*np.array*) – discount rate for every year in years.
- **val_years** (*np.array*) – chash flow at each year.

Return type
float

`climada.util.finance.`**`income_group`**(*cntry_iso*, *ref_year*, *shp_file=None*)

Get country's income group from World Bank's data at a given year, or closest year value. If no data, get the natural earth's approximation.

Parameters

- **cntry_iso** (*str*) – key = ISO alpha_3 country
- **ref_year** (*int*) – reference year
- **shp_file** (*cartopy.io.shapereader.Reader, optional*) – shape file with INCOME_GRP attribute for every country. Load Natural Earth admin0 if not provided.

`climada.util.finance.`**`gdp`**(*cntry_iso*, *ref_year*, *shp_file=None*, *per_capita=False*)

Get country's (current value) GDP from World Bank's data at a given year, or closest year value. If no data, get the natural earth's approximation.

Parameters

- **cntry_iso** (*str*) – key = ISO alpha_3 country
- **ref_year** (*int*) – reference year
- **shp_file** (*cartopy.io.shapereader.Reader, optional*) – shape file with INCOME_GRP attribute for every country. Load Natural Earth admin0 if not provided.
- **per_capita** (*boolean, optional*) – If True, GDP is returned per capita

Return type
float

5.4.11 climada.util.hdf5_handler module

`climada.util.hdf5_handler.`**`read`**(*file_name*, *with_refs=False*)

Load a hdf5 data structure from a file.

Parameters

- **file_name** – file to load
- **with_refs** – enable loading of the references. Default is unset, since it increments the execution time considerably.

Returns
dictionary structure containing all the variables.

Return type
contents

Examples

```
>>> # Contents contains the Matlab data in a dictionary.
>>> contents = read("/pathto/dummy.mat")
>>> # Contents contains the Matlab data and its reference in a dictionary.
>>> contents = read("/pathto/dummy.mat", True)
```

Raises
Exception while reading –

climada.util.hdf5_handler.**get_string**(*array*)
Form string from input array of unisgned integers.

Parameters
array – array of integers

Return type
string

climada.util.hdf5_handler.**get_str_from_ref**(*file_name*, *var*)
Form string from a reference HDF5 variable of the given file.

Parameters

- **file_name** – matlab file name
- **var** – HDF5 reference variable

Return type
string

climada.util.hdf5_handler.**get_list_str_from_ref**(*file_name*, *var*)
Form list of strings from a reference HDF5 variable of the given file.

Parameters

- **file_name** – matlab file name
- **var** – array of HDF5 reference variable

Return type
string

climada.util.hdf5_handler.**get_sparse_csr_mat**(*mat_dict*, *shape*)
Form sparse matrix from input hdf5 sparse matrix data type.

Parameters

- **mat_dict** – dictionary containing the sparse matrix information.
- **shape** – tuple describing output matrix shape.

Return type
sparse csr matrix

5.4.12 climada.util.lines_polys_handler module

class climada.util.lines_polys_handler.**DisaggMethod**(*value*)

Bases: Enum

Disaggregation Method for the … function

DIV : the geometry's distributed to equal parts over all its interpolated points FIX : the geometry's value is replicated over all its interpolated points

DIV = 'div'

FIX = 'fix'

class climada.util.lines_polys_handler.**AggMethod**(*value*)

Bases: Enum

Aggregation Method for the aggregate_impact_mat function

SUM : the impact is summed over all points in the polygon/line

SUM = 'sum'

climada.util.lines_polys_handler.**calc_geom_impact**(*exp*, *impf_set*, *haz*, *res*, *to_meters=False*, *disagg_met=DisaggMethod.DIV*, *disagg_val=None*, *agg_met=AggMethod.SUM*)

Compute impact for exposure with (multi-)polygons and/or (multi-)lines. Lat/Lon values in exp.gdf are ignored, only exp.gdf.geometry is considered.

The geometries are first disaggregated to points. Polygons: grid with resolution res*res. Lines: points along the line separated by distance res. The impact per point is then re-aggregated for each geometry.

Parameters

- **exp** (*Exposures*) – The exposure instance with exp.gdf.geometry containing (multi-)polygons and/or (multi-)lines
- **impf_set** (*ImpactFuncSet*) – The set of impact functions.
- **haz** (*Hazard*) – The hazard instance.
- **res** (*float*) – Resolution of the disaggregation grid (polygon) or line (lines).
- **to_meters** (*bool, optional*) – If True, res is interpreted as meters, and geometries are projected to an equal area projection for disaggregation. The exposures are then projected back to the original projections before impact calculation. The default is False.
- **disagg_met** (*DisaggMethod*) – Disaggregation method of the shapes's original value onto its inter- polated points. 'DIV': Divide the value evenly over all the new points; 'FIX': Replicate the value onto all the new points. Default is 'DIV'. Works in combination with the kwarg 'disagg_val'.
- **disagg_val** (*float, optional*) – Specifies what number should be taken as the value, which is to be disaggregated according to the method provided in disagg_met. None: The shape's value is taken from the exp.gdf.value column. float: This given number will be disaggregated according to the method. In case exp.gdf.value column exists, original values in there will be ignored. The default is None.
- **agg_met** (*AggMethod*) – Aggregation method of the point impacts into impact for respective parent-geometry. If 'SUM', the impact is summed over all points in each geometry. The default is 'SUM'.

Returns
Impact object with the impact per geometry (rows of exp.gdf). Contains two additional attributes 'geom_exp' and 'coord_exp', the first one being the origninal line or polygon geometries for which impact was computed.

Return type
Impact

See also:

exp_geom_to_pnt
disaggregate exposures

`climada.util.lines_polys_handler.`**`impact_pnt_agg`**(*impact_pnt*, *exp_pnt_gdf*, *agg_met*)

Aggregate the impact per geometry.

The output Impact object contains an extra attribute 'geom_exp' containing the geometries.

Parameters

- **impact_pnt** (*Impact*) – Impact object with impact per exposure point (lines of exp_pnt)
- **exp_pnt_gdf** (*gpd.GeoDataFrame*) – Geodataframe of an exposures featuring a multi-index. First level indicating membership of original geometries, second level the disaggregated points. The exposure is obtained for instance with the disaggregation method exp_geom_to_pnt().
- **agg_met** (*AggMethod*) – Aggregation method of the point impacts into impact for respective parent-geometry. If 'SUM', the impact is summed over all points in each geometry. The default is 'SUM'.

Returns
impact_agg – Impact object with the impact per original geometry. Original geometry additionally stored in attribute 'geom_exp'; coord_exp contains only representative points (lat/lon) of those geometries.

Return type
Impact

See also:

exp_geom_to_pnt
exposures disaggregation method

`climada.util.lines_polys_handler.`**`calc_grid_impact`**(*exp*, *impf_set*, *haz*, *grid*, *disagg_met=DisaggMethod.DIV*, *disagg_val=None*, *agg_met=AggMethod.SUM*)

Compute impact for exposure with (multi-)polygons and/or (multi-)lines. Lat/Lon values in exp.gdf are ignored, only exp.gdf.geometry is considered.

The geometries are first disaggregated to points. Polygons: grid with resolution res*res. Lines: points along the line separated by distance res. The impact per point is then re-aggregated for each geometry.

Parameters

- **exp** (*Exposures*) – The exposure instance with exp.gdf.geometry containing (multi-)polygons and/or (multi-)lines
- **impf_set** (*ImpactFuncSet*) – The set of impact functions.
- **haz** (*Hazard*) – The hazard instance.

- **grid** (*np.array()*) – Grid on which to disaggregate the exposures. Provided as two vectors [x_grid, y_grid].
- **disagg_met** (*DisaggMethod*) – Disaggregation method of the shapes's original value onto its inter- polated points. 'DIV': Divide the value evenly over all the new points; 'FIX': Replicate the value onto all the new points. Default is 'DIV'. Works in combination with the kwarg 'disagg_val'.
- **disagg_val** (*float, optional*) – Specifies what number should be taken as the value, which is to be disaggregated according to the method provided in disagg_met. None: The shape's value is taken from the exp.gdf.value column. float: This given number will be disaggregated according to the method. In case exp.gdf.value column exists, original values in there will be ignored The default is None.
- **agg_met** (*AggMethod*) – Aggregation method of the point impacts into impact for respective parent-geometry. If 'SUM', the impact is summed over all points in each geometry. The default is 'SUM'.

Returns
Impact object with the impact per geometry (rows of exp.gdf). Contains two additional attributes 'geom_exp' and 'coord_exp', the first one being the origninal line or polygon geometries for which impact was computed.

Return type
Impact

See also:

`exp_geom_to_pnt`
disaggregate exposures

`climada.util.lines_polys_handler.`**`plot_eai_exp_geom`**(*imp_geom*, *centered=False*, *figsize=(9, 13)*, ***kwargs*)

Plot the average impact per exposure polygon.

Parameters

- **imp_geom** (*Impact*) –

  Impact instance with imp_geom set (i.e. computed from exposures with
  polygons)
- **centered** (*bool, optional*) – Center the plot. The default is False.
- **figsize** (*(float, float), optional*) – Figure size. The default is (9, 13).
- ****kwargs** (*dict*) – Keyword arguments for GeoDataFrame.plot()

Returns
matplotlib axes instance

Return type
ax

`climada.util.lines_polys_handler.`**`exp_geom_to_pnt`**(*exp*, *res*, *to_meters*, *disagg_met*, *disagg_val*)

Disaggregate exposures with (multi-)polygons and/or (multi-)lines geometries to points based on a given resolution.

Parameters

- **exp** (*Exposures*) – The exposure instance with exp.gdf.geometry containing lines or polygons

- **res** (*float*) – Resolution of the disaggregation grid / distance. Can also be a tuple of [x_grid, y_grid] numpy arrays. In this case, to_meters is ignored. This is only possible for Polygon-only exposures.
- **to_meters** (*bool*) – If True, res is interpreted as meters, and geometries are projected to an equal area projection for disaggregation. The exposures are then projected back to the original projections before impact calculation. The default is False.
- **disagg_met** (*DisaggMethod*) – Disaggregation method of the shapes's original value onto its inter- polated points. 'DIV': Divide the value evenly over all the new points; 'FIX': Replicate the value onto all the new points. Default is 'DIV'. Works in combination with the kwarg 'disagg_val'.
- **disagg_val** (*float, optional*) – Specifies what number should be taken as the value, which is to be disaggregated according to the method provided in disagg_met. None: The shape's value is taken from the exp.gdf.value column. float: This given number will be disaggregated according to the method. In case exp.gdf.value column exists, original values in there will be ignored The default is None.

Returns
exp_pnt – Exposures with a double index geodataframe, first level indicating membership of the original geometries of exp, second for the point disaggregation within each geometries.

Return type
Exposures

`climada.util.lines_polys_handler.`**`exp_geom_to_grid`**(*exp*, *grid*, *disagg_met*, *disagg_val*)

Disaggregate exposures with (multi-)polygon geometries to points based on a pre-defined grid.

Parameters

- **exp** (*Exposures*) – The exposure instance with exp.gdf.geometry containing polygons
- **grid** (*np.array()*) – Grid on which to disaggregate the exposures. Provided as two vectors [x_grid, y_grid].
- **disagg_met** (*DisaggMethod*) – Disaggregation method of the shapes's original value onto its inter- polated points. 'DIV': Divide the value evenly over all the new points; 'FIX': Replicate the value onto all the new points. Default is 'DIV'. Works in combination with the kwarg 'disagg_val'.
- **disagg_val** (*float, optional*) – Specifies what number should be taken as the value, which is to be disaggregated according to the method provided in disagg_met. None: The shape's value is taken from the exp.gdf.value column. float: This given number will be disaggregated according to the method. In case exp.gdf.value column exists, original values in there will be ignored The default is None.

Returns
exp_pnt – Exposures with a double index geodataframe, first level indicating membership of the original geometries of exp, second for the point disaggregation within each geometries.

Return type
Exposures

Note: Works with polygon geometries only. No points or lines are allowed.

`climada.util.lines_polys_handler.`**`gdf_to_pnts`**(*gdf*, *res*, *to_meters*)

Disaggregate geodataframe with (multi-)polygons geometries to points.

Parameters

- **gdf** (*gpd.GeoDataFrame*) – Feodataframe instance with gdf.geometry containing (multi)-lines or (multi-)polygons. Points are ignored.
- **res** (*float*) – Resolution of the disaggregation grid. Can also be a tuple of [x_grid, y_grid] numpy arrays. In this case, to_meters is ignored.
- **to_meters** (*bool*) – If True, the geometries are projected to an equal area projection before the disaggregation. res is then in meters. The exposures are then reprojected into the original projections before the impact calculation.

Returns
gdf_pnt – with a double index, first for the geometries of exp, second for the point disaggregation of the geometries.

Return type
gpd.GeoDataFrame

`climada.util.lines_polys_handler.`**`gdf_to_grid`**(*gdf*, *grid*)

Disaggregate geodataframe with (multi-)polygons geometries to points based on a pre-defined grid.

Parameters

- **gdf** (*gpd.GeoDataFrame*) – Geodataframe instance with gdf.geometry containing (multi-)polygons.
- **grid** (*np.array()*) – Grid on which to disaggregate the exposures. Provided as two vectors [x_grid, y_grid].

Returns
gdf_pnt – with a double index, first for the geometries of exp, second for the point disaggregation of the geometries.

Return type
gpd.GeoDataFrame

Note: Works only with polygon geometries. No mixed inputs (with lines or points) are allowed

:raises AttributeError : if other geometry types than polygons are contained in the: :raises dataframe:

`climada.util.lines_polys_handler.`**`reproject_grid`**(*x_grid*, *y_grid*, *orig_crs*, *dest_crs*)

Reproject a grid from one crs to another

Parameters

- **x_grid** – x-coordinates
- **y_grid** – y-coordinates
- **orig_crs** (*pyproj.CRS*) – original CRS of the grid
- **dest_crs** (*pyproj.CRS*) – CRS of the grid to be reprojected to

Returns
Grid coordinates in reprojected crs

Return type
x_trafo, y_trafo

climada.util.lines_polys_handler.**reproject_poly**(*poly*, *orig_crs*, *dest_crs*)

Reproject a polygon from one crs to another

Parameters

- **poly** (*shapely Polygon*) – Polygon
- **orig_crs** (*pyproj.CRS*) – original CRS of the polygon
- **dest_crs** (*pyproj.CRS*) – CRS of the polygon to be reprojected to

Returns
poly – Polygon in desired projection

Return type
shapely Polygon

climada.util.lines_polys_handler.**set_imp_mat**(*impact*, *imp_mat*)

Set Impact attributes from the impact matrix. Returns a copy. Overwrites eai_exp, at_event, aai_agg, imp_mat.

Parameters

- **impact** (*Impact*) – Impact instance.
- **imp_mat** (*sparse.csr_matrix*) – matrix num_events x num_exp with impacts.

Returns
imp – Copy of impact with eai_exp, at_event, aai_agg, imp_mat set.

Return type
Impact

climada.util.lines_polys_handler.**eai_exp_from_mat**(*imp_mat*, *freq*)

Compute impact for each exposures from the total impact matrix

Parameters

- **imp_mat** (*sparse.csr_matrix*) – matrix num_events x num_exp with impacts.
- **frequency** (*np.array*) – frequency of events

Returns
eai_exp – expected impact for each exposure within a period of 1/frequency_unit

Return type
np.array

climada.util.lines_polys_handler.**at_event_from_mat**(*imp_mat*)

Compute impact for each hazard event from the total impact matrix

Parameters
imp_mat (*sparse.csr_matrix*) – matrix num_events x num_exp with impacts.

Returns
at_event – impact for each hazard event

Return type
np.array

climada.util.lines_polys_handler.**aai_agg_from_at_event**(*at_event*, *freq*)

Aggregate impact.at_event

Parameters

- **at_event** (*np.array*) – impact for each hazard event

- **frequency** (*np.array*) – frequency of event

Returns
average impact within a period of 1/frequency_unit, aggregated

Return type
float

5.4.13 climada.util.plot module

climada.util.plot.**geo_bin_from_array**(*array_sub*, *geo_coord*, *var_name*, *title*, *pop_name=True*, *buffer=1.0*, *extend='neither'*, *proj=<Derived Projected CRS: +proj=eqc +ellps=WGS84 +a=6378137.0 +lon_0=0.0 +to ...> Name: unknown Axis Info [cartesian]: - E[east]: Easting (unknown) - N[north]: Northing (unknown) - h[up]: Ellipsoidal height (metre) Area of Use: - undefined Coordinate Operation: - name: unknown - method: Equidistant Cylindrical Datum: unknown - Ellipsoid: WGS 84 - Prime Meridian: Greenwich*, *shapes=True*, *axes=None*, *figsize=(9, 13)*, *adapt_fontsize=True*, ***kwargs*)

Plot array values binned over input coordinates.

Parameters

- **array_sub** (*np.array(1d or 2d) or list(np.array)*) – Each array (in a row or in the list) are values at each point in corresponding geo_coord that are binned in one subplot.
- **geo_coord** (*2d np.array or list(2d np.array)*) – (lat, lon) for each point in a row. If one provided, the same grid is used for all subplots. Otherwise provide as many as subplots in array_sub.
- **var_name** (*str or list(str)*) – label to be shown in the colorbar. If one provided, the same is used for all subplots. Otherwise provide as many as subplots in array_sub.
- **title** (*str or list(str)*) – subplot title. If one provided, the same is used for all subplots. Otherwise provide as many as subplots in array_sub.
- **pop_name** (*bool, optional*) – add names of the populated places, by default True.
- **buffer** (*float, optional*) – border to add to coordinates, by default BUFFER
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max'], by default 'neither'
- **proj** (*ccrs, optional*) – coordinate reference system of the given data, by default ccrs.PlateCarree()
- **shapes** (*bool, optional*) – Overlay Earth's countries coastlines to matplotlib.pyplot axis. The default is True
- **axes** (*Axes or ndarray(Axes), optional*) – by default None
- **figsize** (*tuple, optional*) – figure size for plt.subplots, by default (9, 13)
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- ****kwargs** – arbitrary keyword arguments for hexbin matplotlib function

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

Raises
`ValueError:` – Input array size missmatch

`climada.util.plot.`**`geo_im_from_array`**(*array_sub*, *coord*, *var_name*, *title*, *proj=None*, *smooth=True*, *shapes=True*, *axes=None*, *figsize=(9, 13)*, *adapt_fontsize=True*, ***kwargs*)

Image(s) plot defined in array(s) over input coordinates.

Parameters

- **array_sub** (*np.array(1d or 2d) or list(np.array)*) – Each array (in a row or in the list) are values at each point in corresponding geo_coord that are ploted in one subplot.
- **coord** (*2d np.array*) – (lat, lon) for each point in a row. The same grid is used for all subplots.
- **var_name** (*str or list(str)*) – label to be shown in the colorbar. If one provided, the same is used for all subplots. Otherwise provide as many as subplots in array_sub.
- **title** (*str or list(str)*) – subplot title. If one provided, the same is used for all subplots. Otherwise provide as many as subplots in array_sub.
- **proj** (*ccrs, optional*) – coordinate reference system used in coordinates, by default None
- **smooth** (*bool, optional*) – smooth plot to RESOLUTIONxRESOLUTION, by default True
- **shapes** (*bool, optional*) – Overlay Earth's countries coastlines to matplotlib.pyplot axis. The default is True
- **axes** (*Axes or ndarray(Axes), optional*) – by default None
- **figsize** (*tuple, optional*) – figure size for plt.subplots, by default (9, 13)
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- ****kwargs** – arbitrary keyword arguments for pcolormesh matplotlib function

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

Raises
`ValueError` –

`climada.util.plot.`**`make_map`**(*num_sub=1*, *figsize=(9, 13)*, *proj=<Derived Projected CRS: +proj=eqc +ellps=WGS84 +a=6378137.0 +lon_0=0.0 +to ...> Name: unknown Axis Info [cartesian]: - E[east]: Easting (unknown) - N[north]: Northing (unknown) - h[up]: Ellipsoidal height (metre) Area of Use: - undefined Coordinate Operation: - name: unknown - method: Equidistant Cylindrical Datum: unknown - Ellipsoid: WGS 84 - Prime Meridian: Greenwich*, *adapt_fontsize=True*)

Create map figure with cartopy.

Parameters

- **num_sub** (*int or tuple*) – number of total subplots in figure OR number of subfigures in row and column: (num_row, num_col).
- **figsize** (*tuple*) – figure size for plt.subplots
- **proj** (*cartopy.crs projection, optional*) – geographical projection, The default is PlateCarree default.
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.

Returns
fig, axis_sub, fontsize

Return type
matplotlib.figure.Figure, cartopy.mpl.geoaxes.GeoAxesSubplot, int

`climada.util.plot.`**`add_shapes`**(*axis*)

Overlay Earth's countries coastlines to matplotlib.pyplot axis.

Parameters

- **axis** (*cartopy.mpl.geoaxes.GeoAxesSubplot*) – Cartopy axis
- **projection** (*cartopy.crs projection, optional*) – Geographical projection. The default is PlateCarree.

`climada.util.plot.`**`add_populated_places`**(*axis*, *extent*, *proj=<Derived Projected CRS: +proj=eqc +ellps=WGS84 +a=6378137.0 +lon_0=0.0 +to ...> Name: unknown Axis Info [cartesian]: - E[east]: Easting (unknown) - N[north]: Northing (unknown) - h[up]: Ellipsoidal height (metre) Area of Use: - undefined Coordinate Operation: - name: unknown - method: Equidistant Cylindrical Datum: unknown - Ellipsoid: WGS 84 - Prime Meridian: Greenwich*, *fontsize=None*)

Add city names.

Parameters

- **axis** (*cartopy.mpl.geoaxes.GeoAxesSubplot*) – cartopy axis.
- **extent** (*list*) – geographical limits [min_lon, max_lon, min_lat, max_lat]
- **proj** (*cartopy.crs projection, optional*) – geographical projection, The default is PlateCarree.
- **fontsize** (*int, optional*) – Size of the fonts. If set to None, the default matplotlib settings are used.

`climada.util.plot.`**`add_cntry_names`**(*axis*, *extent*, *proj=<Derived Projected CRS: +proj=eqc +ellps=WGS84 +a=6378137.0 +lon_0=0.0 +to ...> Name: unknown Axis Info [cartesian]: - E[east]: Easting (unknown) - N[north]: Northing (unknown) - h[up]: Ellipsoidal height (metre) Area of Use: - undefined Coordinate Operation: - name: unknown - method: Equidistant Cylindrical Datum: unknown - Ellipsoid: WGS 84 - Prime Meridian: Greenwich*, *fontsize=None*)

Add country names.

Parameters

- **axis** (*cartopy.mpl.geoaxes.GeoAxesSubplot*) – Cartopy axis.
- **extent** (*list*) – geographical limits [min_lon, max_lon, min_lat, max_lat]
- **proj (*cartopy.crs projection, optional*) – Geographical projection.**
  The default is PlateCarree.

  fontsize
  [int, optional] Size of the fonts. If set to None, the default matplotlib settings are used.

5.4.14 climada.util.save module

`climada.util.save.`**`save`**(*out_file_name*, *var*)

Save variable with provided file name. Uses configuration save_dir folder if no absolute path provided.

Parameters

- **out_file_name** (*str*) – file name (absolute path or relative to configured save_dir)
- **var** (*object*) – variable to save in pickle format

`climada.util.save.`**`load`**(*in_file_name*)

Load variable contained in file. Uses configuration save_dir folder if no absolute path provided.

Parameters
in_file_name (*str*) – file name

Return type
object

5.4.15 climada.util.scalebar_plot module

`climada.util.scalebar_plot.`**`scale_bar`**(*ax*, *location*, *length*, *metres_per_unit=1000*, *unit_name='km'*, *tol=0.01*, *angle=0*, *color='black'*, *linewidth=3*, *text_offset=0.005*, *ha='center'*, *va='bottom'*, *plot_kwargs=None*, *text_kwargs=None*, ***kwargs*)

Add a scale bar to CartoPy axes.

For angles between 0 and 90 the text and line may be plotted at slightly different angles for unknown reasons. To work around this, override the 'rotation' keyword argument with text_kwargs.

Parameters

- **ax** – CartoPy axes.
- **location** – Position of left-side of bar in axes coordinates.
- **length** – Geodesic length of the scale bar.
- **metres_per_unit** – Number of metres in the given unit. Default: 1000
- **unit_name** – Name of the given unit. Default: 'km'
- **tol** – Allowed relative error in length of bar. Default: 0.01
- **angle** – Anti-clockwise rotation of the bar.
- **color** – Color of the bar and text. Default: 'black'
- **linewidth** – Same argument as for plot.
- **text_offset** – Perpendicular offset for text in axes coordinates. Default: 0.005
- **ha** – Horizontal alignment. Default: 'center'
- **va** – Vertical alignment. Default: 'bottom'
- **plot_kwargs** – Keyword arguments for plot, overridden by ***kwargs*.
- **text_kwargs** – Keyword arguments for text, overridden by ***kwargs*.
- **kwargs** – Keyword arguments for both plot and text.

5.4.16 climada.util.select module

climada.util.select.**get_attributes_with_matching_dimension**(*obj*, *dims*)

Get the attributes of an object that have len(dims) number of dimensions or more, and all dims are individual parts of the attribute's shape.

Parameters

- **obj** (*object of any class*) – The object from which matching attributes are returned
- **dims** (*list[int]*) – List of dimensions size to match

Returns
list_of_attrs – List of names of the attributes with matching dimensions

Return type
list[str]

5.4.17 climada.util.value_representation module

climada.util.value_representation.**sig_dig**(*x*, *n_sig_dig=16*)

Rounds x to n_sig_dig number of significant digits. 0, inf, Nan are returned unchanged.

Examples

with n_sig_dig = 5:

1.234567 -> 1.2346, 123456.89 -> 123460.0

Parameters

- **x** (*float*) – number to be rounded
- **n_sig_dig** (*int, optional*) – Number of significant digits. The default is 16.

Returns
Rounded number

Return type
float

climada.util.value_representation.**sig_dig_list**(*iterable*, *n_sig_dig=16*)

Vectorized form of sig_dig. Rounds a list of float to a number of significant digits

Parameters

- **iterable** (*iter(float)*) – iterable of numbers to be rounded
- **n_sig_dig** (*int, optional*) – Number of significant digits. The default is 16.

Returns
list of rounded floats

Return type
list

climada.util.value_representation.**convert_monetary_value**(*values*, *abbrev*, *n_sig_dig=None*) climada.util.value_representation.**value_to_monetary_unit**(*values*, *n_sig_dig=None*, *abbreviations=None*)

Converts list of values to closest common monetary unit.

0, Nan and inf have not unit.

Parameters

- **values** (*int or float, list(int or float) or np.ndarray(int or float)*) – Values to be converted
- **n_sig_dig** (*int, optional*) – Number of significant digits to return.

  Examples: n_sig_di=5: 1.234567 -> 1.2346, 123456.89 -> 123460.0

  Default: all digits are returned.
- **abbreviations** (*dict, optional*) – Name of the abbreviations for the money 1000s counts

  Default: { 1:", 1000: 'K', 1000000: 'M', 1000000000: 'Bn', 1000000000000: 'Tn' }

Returns

- **mon_val** (*np.ndarray*) – Array of values in monetary unit
- **name** (*string*) – Monetary unit

Examples

values = [1e6, 2*1e6, 4.5*1e7, 0, Nan, inf] ->
[1, 2, 4.5, 0, Nan, inf] ['M']

climada.util.value_representation.**safe_divide**(*numerator*, *denominator*, *replace_with=nan*)

Safely divide two arrays or scalars.

This function handles division by zero and NaN values in the numerator or denominator on an element-wise basis. If the division results in infinity, NaN, or division by zero in any element, that particular result is replaced by the specified value.

Parameters

- **numerator** (*np.ndarray or scalar*) – The numerator for division.
- **denominator** (*np.ndarray or scalar*) – The denominator for division. Division by zero and NaN values are handled safely.
- **replace_with** (*float, optional*) – The value to use in place of division results that are infinity, NaN, or division by zero. By default, it is NaN.

Returns

The result of the division. If the division results in infinity, NaN, or division by zero in any element, it returns the value specified in `replace_with` for those elements.

Return type

np.ndarray or scalar

Notes

The function uses numpy's `true_divide` for array-like inputs and handles both scalar and array-like inputs for the numerator and denominator. NaN values or division by zero in any element of the input will result in the *replace_with* value in the corresponding element of the output.

Examples

```
>>> safe_divide(1, 0)
nan
```

```
>>> safe_divide(1, 0, replace_with=0)
0
```

```
>>> safe_divide([1, 0, 3], [0, 0, 3])
array([nan, nan,  1.])
```

```
>>> safe_divide([4, 4], [1, 0])
array([4., nan])
```

```
>>> safe_divide([4, 4], [1, nan])
array([ 4., nan])
```

5.4.18 climada.util.yearsets module

`climada.util.yearsets.`**`impact_yearset`**(*imp*, *sampled_years*, *lam=None*, *correction_fac=True*, *seed=None*)

Create a yearset of impacts (yimp) containing a probabilistic impact for each year in the sampled_years list by sampling events from the impact received as input with a Poisson distribution centered around lam per year (lam = sum(imp.frequency)). In contrast to the expected annual impact (eai) yimp contains impact values that differ among years. When correction factor is true, the yimp are scaled such that the average over all years is equal to the eai.

Parameters

- **imp** (*climada.engine.Impact()*) – impact object containing impacts per event
- **sampled_years** (*list*) – A list of years that shall be covered by the resulting yimp.
- **seed** (*Any, optional*) – seed for the default bit generator default: None

Optional parameters

lam: int
The applied Poisson distribution is centered around lam events per year. If no lambda value is given, the default lam = sum(imp.frequency) is used.

correction_fac
[boolean] If True a correction factor is applied to the resulting yimp. It is scaled in such a way that the expected annual impact (eai) of the yimp equals the eai of the input impact

Returns

- **yimp** (*climada.engine.Impact()*) – yearset of impacts containing annual impacts for all sampled_years
- **sampling_vect** (*2D array*) – The sampling vector specifies how to sample the yimp, it consists of one sub-array per sampled_year, which contains the event_ids of the events used to calculate the annual impacts. Can be used to re-create the exact same yimp.

`climada.util.yearsets.`**`impact_yearset_from_sampling_vect`**(*imp*, *sampled_years*, *sampling_vect*, *correction_fac=True*)

Create a yearset of impacts (yimp) containing a probabilistic impact for each year in the sampled_years list by sampling events from the impact received as input following the sampling vector provided. In contrast to the expected annual impact (eai) yimp contains impact values that differ among years. When correction factor is true, the yimp are scaled such that the average over all years is equal to the eai.

Parameters

- **imp** (*climada.engine.Impact()*) – impact object containing impacts per event
- **sampled_years** (*list*) – A list of years that shall be covered by the resulting yimp.
- **sampling_vect** (*2D array*) – The sampling vector specifies how to sample the yimp, it consists of one sub-array per sampled_year, which contains the event_ids of the events used to calculate the annual impacts. It needs to be obtained in a first call, i.e. [yimp, sampling_vect] = climada_yearsets.impact_yearset(…) and can then be provided in this function to obtain the exact same sampling (also for a different imp object)

Optional parameter

correction_fac
[boolean] If True a correction factor is applied to the resulting yimp. It is scaled in such a way that the expected annual impact (eai) of the yimp equals the eai of the input impact

Returns
yimp – yearset of impacts containing annual impacts for all sampled_years

Return type
climada.engine.Impact()

`climada.util.yearsets.`**`sample_from_poisson`**(*n_sampled_years*, *lam*, *seed=None*)

Sample the number of events for n_sampled_years

Parameters

- **n_sampled_years** (*int*) – The target number of years the impact yearset shall contain.
- **lam** (*float*) – the applied Poisson distribution is centered around lambda events per year
- **seed** (*int, optional*) – seed for numpy.random, will be set if not None default: None

Returns
events_per_year – Number of events per sampled year

Return type
np.ndarray

`climada.util.yearsets.`**`sample_events`**(*events_per_year*, *freqs_orig*, *seed=None*)

Sample events uniformely from an array (indices_orig) without replacement (if sum(events_per_year) > n_input_events the input events are repeated (tot_n_events/n_input_events) times, by ensuring that the same events doens't occur more than once per sampled year).

Parameters

- **events_per_year** (*np.ndarray*) – Number of events per sampled year
- **freqs_orig** (*np.ndarray*) – Frequency of each input event
- **seed** (*Any, optional*) – seed for the default bit generator. Default: None

Returns
sampling_vect – The sampling vector specifies how to sample the yimp, it consists of one sub-array per sampled_year, which contains the event_ids of the events used to calculate the annual impacts.

Return type
2D array

`climada.util.yearsets.`**`compute_imp_per_year`**(*imp*, *sampling_vect*)

Sample annual impacts from the given event_impacts according to the sampling dictionary

Parameters

- **imp** (*climada.engine.Impact()*) – impact object containing impacts per event
- **sampling_vect** (*2D array*) – The sampling vector specifies how to sample the yimp, it consists of one sub-array per sampled_year, which contains the event_ids of the events used to calculate the annual impacts.

Returns
imp_per_year – Sampled impact per year (length = sampled_years)

Return type
np.ndarray

`climada.util.yearsets.`**`calculate_correction_fac`**(*imp_per_year*, *imp*)

Calculate a correction factor that can be used to scale the yimp in such a way that the expected annual impact (eai) of the yimp amounts to the eai of the input imp

Parameters

- **imp_per_year** (*np.ndarray*) – sampled yimp
- **imp** (*climada.engine.Impact()*) – impact object containing impacts per event

Returns
correction_factor – The correction factor is calculated as imp_eai/yimp_eai

Return type
int

CHAPTER

SIX

CLIMADA OVERVIEW

6.1 Contents

6.2 Introduction

6.2.1 What is CLIMADA?

CLIMADA is a fully probabilistic climate risk assessment tool. It provides a framework for users to combine exposure, hazard and vulnerability or impact data to calculate risk. Users can create probabilistic impact data from event sets, look at how climate change affects these impacts, and see how effectively adaptation measures can change them. CLIMADA also allows for studies of individual events, historical event sets and forecasts.

The model is highly customisable, meaning that users can work with out-of-the-box data provided for different hazards, population and economic exposure, or can provide their own data for part or all of the analysis. The pre-packaged data make CLIMADA particularly useful for users who focus on just one element of risk, since CLIMADA can 'fill in the gaps' for hazard, exposure or vulnerability in the rest of the analysis.

The model core is designed to give as much flexibility as possible when describing the elements of risk, meaning that CLIMADA isn't limited to particular hazards, exposure types or impacts. We love to see the model applied to new problems and contexts.

CLIMADA provides classes, methods and data for exposure, hazard and impact functions (also called vulnerability functions), plus a financial model and a framework to analyse adaptation measures. Additional classes and data for common uses, such as economic exposures or tropical storms and tutorials for every class are available: see the *CLIMADA features* section below.

6.2.2 This tutorial

This tutorial is for people new to CLIMADA who want to get a high level understanding of the model and work through an example risk analysis. It will list the current features of the model, and go through a complete CLIMADA analysis to give an idea of how the model works. Other tutorials go into more detail about different model components and individual hazards.

6.2.3 Resources beyond this tutorial

- *Installation guide* - go here if you've not installed the model yet
- CLIMADA Read the Docs home page - for all other documentation
- *List of CLIMADA's features and associated tutorials*
- CLIMADA GitHub develop branch documentation for the very latest versions of code and documentation
- CLIMADA paper GitHub repository - for publications using CLIMADA

6.3 CLIMADA features

A risk analysis with CLIMADA can include

1. the statistical risk to your exposure from a set of events,
2. how it changes under climate change, and
3. a cost-benefit analysis of adaptation measures.

CLIMADA is flexible: the "statistical risk" above could be describing the annual expected insured flood losses to a property portfolio, the number of people displaced by an ensemble of typhoon forecasts, the annual disruption to a railway network from landslides, or changes to crop yields.

Users from risk-analysis backgrounds will be familiar with describing the impact of events by combining exposure, hazard and an impact function (or vulnerability curve) that combines the two to describe a hazard's effects. A CLIMADA analysis uses the same approach but wraps the exposures and their impact functions into a single `Entity` class, along with discount rates and adaptation options (see the below tutorials for more on CLIMADA's financial model).

CLIMADA's `Impact` object is used to analyse events and event sets, whether this is the impact of a single wildfire, or the global economic risk from tropical cyclones in 2100.

CLIMADA is divided into two parts (two repositories):

1. the core climada_python contains all the modules necessary for the probabilistic impact, the averted damage, uncertainty and forecast calculations. Data for hazard, exposures and impact functions can be obtained from the data API. Litpop is included as demo Exposures module, and Tropical cyclones is included as a demo Hazard module.
2. the petals climada_petals contains all the modules for generating data (e.g., TC_Surge, WildFire, OpenStreeMap, …). Most development is done here. The petals builds-upon the core and does not work as a stand-alone.

6.3.1 CLIMADA classes

This is a full directory of tutorials for CLIMADA's classes to use as a reference. You don't need to read all this to do this tutorial, but it may be useful to refer back to.

Core (climada_python):

- ***Hazard***: a class that stores sets of geographic hazard footprints, (e.g. for wind speed, water depth and fraction, drought index), and metadata including event frequency. Several predefined extensions to create particular hazards from particular datasets and models are included with CLIMADA:
  - *Tropical cyclone wind*: global hazard sets for tropical cyclone events, constructing statistical wind fields from storm tracks. Subclasses include methods and data to calculate historical wind footprints, create forecast enembles from ECMWF tracks, and create climatological event sets for different climate scenarios.
  - *European windstorms*: includes methods to read and plot footprints from the Copernicus WISC dataset and for DWD and ICON forecasts.
- ***Entity***: this is a container that groups CLIMADA's socio-economic models. It's is where the Exposures and Impact Functions are stored, which can then be combined with a hazard for a risk analysis (using the Engine's Impact class). It is also where Discount Rates and Measure Sets are stored, which are used in adaptation cost-benefit analyses (using the Engine's CostBenefit class):
  - *Exposures*: geolocated exposures. Each exposure is associated with a value (which can be a dollar value, population, crop yield, etc), information to associate it with impact functions for the relevant hazard(s) (in the Entity's ImpactFuncSet), a geometry, and other optional properties such as deductables and cover. Exposures can be loaded from a file, specified by the user, or created from regional economic models accessible within CLIMADA, for example:
    - *LitPop*: regional economic model using nightlight and population maps together with several economic indicators
    - *Polygons_lines*: use CLIMADA Impf you have your exposure in the form of shapes/polygons or in the form of lines.
  - *ImpactFuncSet*: functions to describe the impacts that hazards have on exposures, expressed in terms of e.g. the % dollar value of a building lost as a function of water depth, or the mortality rate for over-70s as a function of temperature. CLIMADA provides some common impact functions, or they can be user-specified. The following is an incomplete list:
    - ImpactFunc: a basic adjustable impact function, specified by the user
    - IFTropCyclone: impact functions for tropical cyclone winds

    * IFRiverFlood: impact functions for river floods
    * IFStormEurope: impact functions for European windstorms

  - *DiscRates*: discount rates per year
  - *MeasureSet*: a collection of Measure objects that together describe any adaptation measures being modelled. Adaptation measures are described by their cost, and how they modify exposure, hazard, and impact functions (and have have a method to do these things). Measures also include risk transfer options.

- ***Engine***: the CLIMADA Engine contains the Impact and CostBenefit classes, which are where the main model calculations are done, combining Hazard and Entity objects.
  - *Impact*: a class that stores CLIMADA's modelled impacts and the methods to calculate them from Exposure, Impact Function and Hazard classes. The calculations include average annual impact, expected annual impact by exposure item, total impact by event, and (optionally) the impact of each event on each exposure point. Includes statistical and plotting routines for common analysis products.
  - *Impact_data*: The core functionality of the module is to read disaster impact data as downloaded from the International Disaster Database EM-DAT (www.emdat.be) and produce a CLIMADA Impact()-instance from it. The purpose is to make impact data easily available for comparison with simulated impact inside CLIMADA, e.g. for calibration purposes.
  - *CostBenefit*: a class to appraise adaptation options. It uses an Entity's MeasureSet to calculate new Impacts based on their adjustments to hazard, exposure, and impact functions, and returns statistics and plotting routines to express cost-benefit comparisons.
  - *Unsequa*: a module for uncertainty and sensitivity analysis.
  - *Unsequa_helper*: The InputVar class provides a few helper methods to generate generic uncertainty input variables for exposures, impact function sets, hazards, and entities (including measures cost and disc rates). This tutorial complements the general tutorial on the uncertainty and sensitivity analysis module unsequa.
  - *Forecast*: This class deals with weather forecasts and uses CLIMADA ImpactCalc.impact() to forecast impacts of weather events on society. It mainly does one thing: It contains all plotting and other functionality that are specific for weather forecasts, impact forecasts and warnings.

climada_petals:

- ***Hazard***:
  - Storm surge: Tropical cyclone surge from linear wind-surge relationship and a bathtub model.
  - River flooding: global water depth hazard for flood, including methods to work with ISIMIP simulations.
  - Crop modelling: combines ISIMIP crop simulations and UN Food and Agrigultre Organization data. The module uses crop production as exposure, with hydrometeorological 'hazard' increasing or decreasing production.
  - Wildfire (global): This class is used to model the wildfire hazard using the historical data available and creating synthetic fires which are summarized into event years to establish a comprehensiv probabilistic risk assessment.
  - Landslide: This class is able to handle two different types of landslide source files (in one case, already the finished product of some model output, in the other case just a historic data collection).
  - TCForecast: This class extends the TCTracks class with methods to download operational ECMWF ensemble tropical storm track forecasts, read the BUFR files they're contained in and produce a TCTracks object that can be used to generate TropCyclone hazard footprints.
  - Emulator:Given a database of hazard events, this module climada.hazard.emulator provides tools to subsample events (or time series of events) from that event database.
  - Drought (global): tutorial under development

- ***Entity***:
  - *Exposures*:
    - BlackMarble: regional economic model from nightlight intensities and economic indicators (GDP, income group). Largely succeeded by LitPop.
    - OpenStreetMap: CLIMADA provides some ways to make use of the entire OpenStreetMap data world and to use those data within the risk modelling chain of CLIMADA as exposures.
- ***Engine***:
  - SupplyChain: This class allows assessing indirect impacts via Input-Ouput modeling.

This list will be updated periodically along with new CLIMADA releases. To see the latest, development version of all tutorials, see the tutorials page on the CLIMADA GitHub.

6.4 Tutorial: an example risk assessment

This example will work through a risk assessment for tropical storm wind in Puerto Rico, constructing hazard, exposure and vulnerability and combining them to create an Impact object. Everything you need for this is included in the main CLIMADA installation and additional data will be downloaded by the scripts as required.

6.5 Hazard

Hazards are characterized by their frequency of occurrence and the geographical distribution of their intensity. The `Hazard` class collects events of the same hazard type (e.g. tropical cyclone, flood, drought, …) with intensity values over the same geographic centroids. They might be historical events or synthetic.

See the *Hazard tutorial* to learn about the Hazard class in more detail, and the *CLIMADA features* section of this document to explore tutorials for different hazards, including *tropical cyclones*, as used here.

Tropical cyclones in CLIMADA and the `TropCyclone` class work like any hazard, storing each event's wind speeds at the geographic centroids specified for the class. Pre-calculated hazards can be loaded from files (see the *full Hazard tutorial*, but they can also be modelled from a storm track using the `TCTracks` class, based on a storm's parameters at each time step. This is how we'll construct the hazards for our example.

So before we create the hazard, we will create our storm tracks and define the geographic centroids for the locations we want to calculate hazard at.

6.5.1 Storm tracks

Storm tracks are created and stored in a separate class, `TCTracks`. We use its method `from_ibtracs_netcdf` to create the tracks from the IBTRaCS storm tracks archive. In the next block we will download the full dataset, which might take a little time. However, to plot the whole dataset takes too long (see the second block), so we choose a shorter time range here to show the function. See the *full TropCyclone tutorial* for more detail and troubleshooting.

```
import numpy as np
from climada.hazard import TCTracks
import warnings # To hide the warnings
warnings.filterwarnings('ignore')

tracks = TCTracks.from_ibtracs_netcdf(provider='usa', basin='NA') # Here we download␣
↪the full dataset for the analysis
```

(continues on next page)

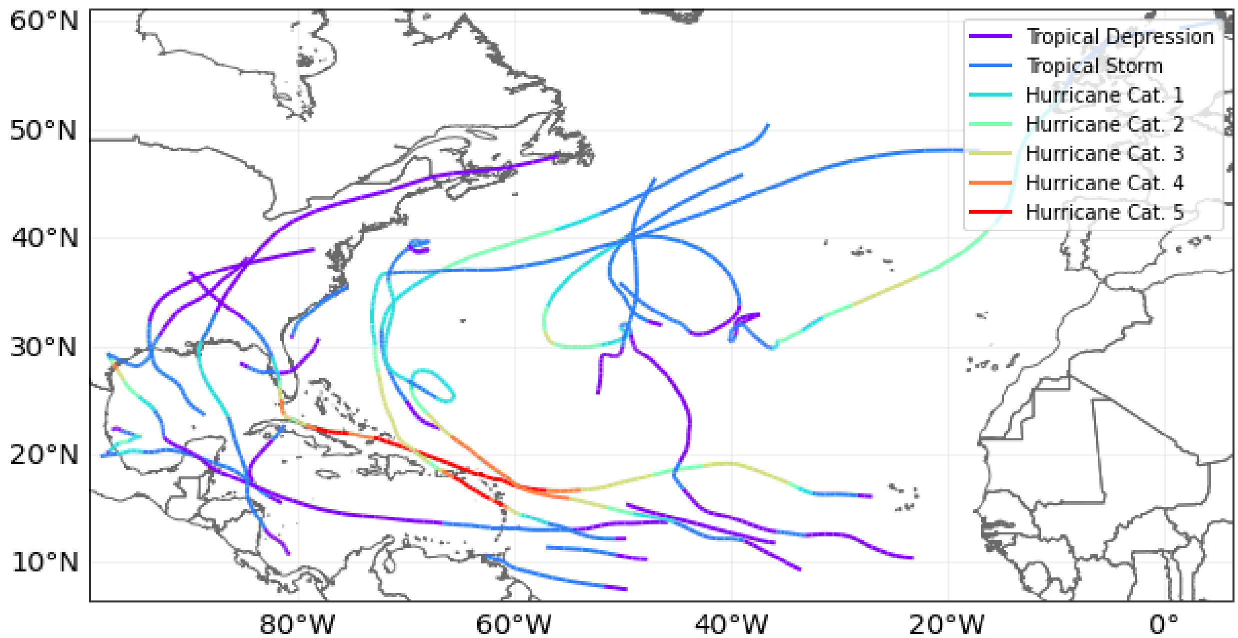
(continued from previous page)

```
# afterwards (e.g. return period), but you can also use "year_range" to adjust the␣
↪range of the dataset to be downloaded.
# While doing that, you need to make sure that the year 2017 is included if you want␣
↪to run the blocks with the codes
# subsetting a specific tropic cyclone, which happened in 2017. (Of course, you can␣
↪also change the subsetting codes.)
```

```
2022-03-21 14:31:20,322 - climada.hazard.tc_tracks - WARNING - 1122 storm events are␣
↪discarded because no valid wind/pressure values have been found: 1851175N26270,␣
↪1851181N19275, 1851187N22262, 1851192N12300, 1851214N14321, ...
2022-03-21 14:31:20,345 - climada.hazard.tc_tracks - WARNING - 139 storm events are␣
↪discarded because only one valid timestep has been found: 1852232N21293,␣
↪1853242N12336, 1855236N12304, 1856221N25277, 1856235N13302, ...
2022-03-21 14:31:22,766 - climada.hazard.tc_tracks - INFO - Progress: 10%
2022-03-21 14:31:25,059 - climada.hazard.tc_tracks - INFO - Progress: 20%
2022-03-21 14:31:27,491 - climada.hazard.tc_tracks - INFO - Progress: 30%
2022-03-21 14:31:30,067 - climada.hazard.tc_tracks - INFO - Progress: 40%
2022-03-21 14:31:32,415 - climada.hazard.tc_tracks - INFO - Progress: 50%
2022-03-21 14:31:34,829 - climada.hazard.tc_tracks - INFO - Progress: 60%
2022-03-21 14:31:37,482 - climada.hazard.tc_tracks - INFO - Progress: 70%
2022-03-21 14:31:39,976 - climada.hazard.tc_tracks - INFO - Progress: 80%
2022-03-21 14:31:42,307 - climada.hazard.tc_tracks - INFO - Progress: 90%
2022-03-21 14:31:44,580 - climada.hazard.tc_tracks - INFO - Progress: 100%
2022-03-21 14:31:45,780 - climada.hazard.tc_tracks - INFO - Progress: 10%
2022-03-21 14:31:45,833 - climada.hazard.tc_tracks - INFO - Progress: 21%
2022-03-21 14:31:45,886 - climada.hazard.tc_tracks - INFO - Progress: 31%
2022-03-21 14:31:45,939 - climada.hazard.tc_tracks - INFO - Progress: 42%
2022-03-21 14:31:45,992 - climada.hazard.tc_tracks - INFO - Progress: 52%
2022-03-21 14:31:46,048 - climada.hazard.tc_tracks - INFO - Progress: 63%
2022-03-21 14:31:46,100 - climada.hazard.tc_tracks - INFO - Progress: 73%
2022-03-21 14:31:46,150 - climada.hazard.tc_tracks - INFO - Progress: 84%
2022-03-21 14:31:46,203 - climada.hazard.tc_tracks - INFO - Progress: 94%
2022-03-21 14:31:46,232 - climada.hazard.tc_tracks - INFO - Progress: 100%
```

This will load all historical tracks in the North Atlantic into the `tracks` object (since we set `basin='NA'`). The `TCTracks.plot` method will plot the downloaded tracks, though there are too many for the plot to be very useful:

```
# plotting tracks can be very time consuming, depending on the number of tracks. So␣
↪we choose only a few here, by limiting the time range to one year
tracks_2017 = TCTracks.from_ibtracs_netcdf(provider='usa', basin='NA', year_range =␣
↪(2017, 2017))
tracks_2017 .plot(); # This may take a very long time
```



It's also worth adding additional time steps to the tracks (though this can be memory intensive!). Most tracks are reported at 3-hourly intervals (plus a frame at landfall). Event footprints are calculated as the maximum wind from any time step. For a fast-moving storm these combined three-hourly footprints give quite a rough event footprint, and it's worth adding extra frames to smooth the footprint artificially (try running this notebook with and without this interpolation to see the effect):

```
tracks.equal_timestep(time_step_h=0.5)
```

```
2022-03-21 14:32:39,466 - climada.hazard.tc_tracks - INFO - Interpolating 1049 tracks␣
↪to 0.5h time steps.
```

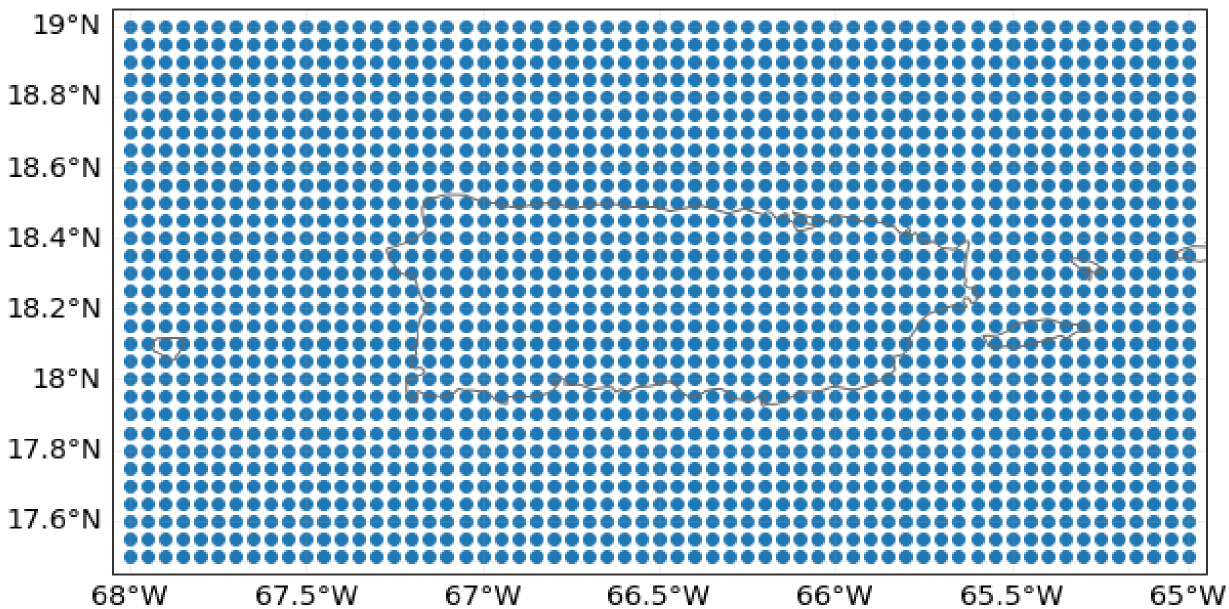
Now, irresponsibly for a risk analysis, we're only going to use these historical events: they're enough to demonstrate CLIMADA in action. A proper risk analysis would expand it to include enough events for a statistically robust climatology. See the *full TropCyclone tutorial* for CLIMADA's stochastic event generation.

6.5.2 Centroids

A hazard's centroids can be any set of locations where we want the hazard to be evaluated. This could be the same as the locations of your exposure, though commonly it is on a regular lat-lon grid (with hazard being imputed to exposure between grid points).

Here we'll set the centroids as a 0.1 degree grid covering Puerto Rico. Centroids are defined by a `Centroids` class, which has the `from_pnt_bounds` method for generating regular grids and a `plot` method to inspect the centroids.

```
from climada.hazard import Centroids

min_lat, max_lat, min_lon, max_lon = 17.5, 19.0, -68.0, -65.0
cent = Centroids.from_pnt_bounds((min_lon, min_lat, max_lon, max_lat), res=0.05)
cent.check()
cent.plot();
```



Almost every class in CLIMADA has a `check()` method, as used above. This verifies that the necessary data for an object is correctly provided and logs the optional variables that are not present. It is always worth running it after filling an instance of an object.

6.5.3 Hazard footprint

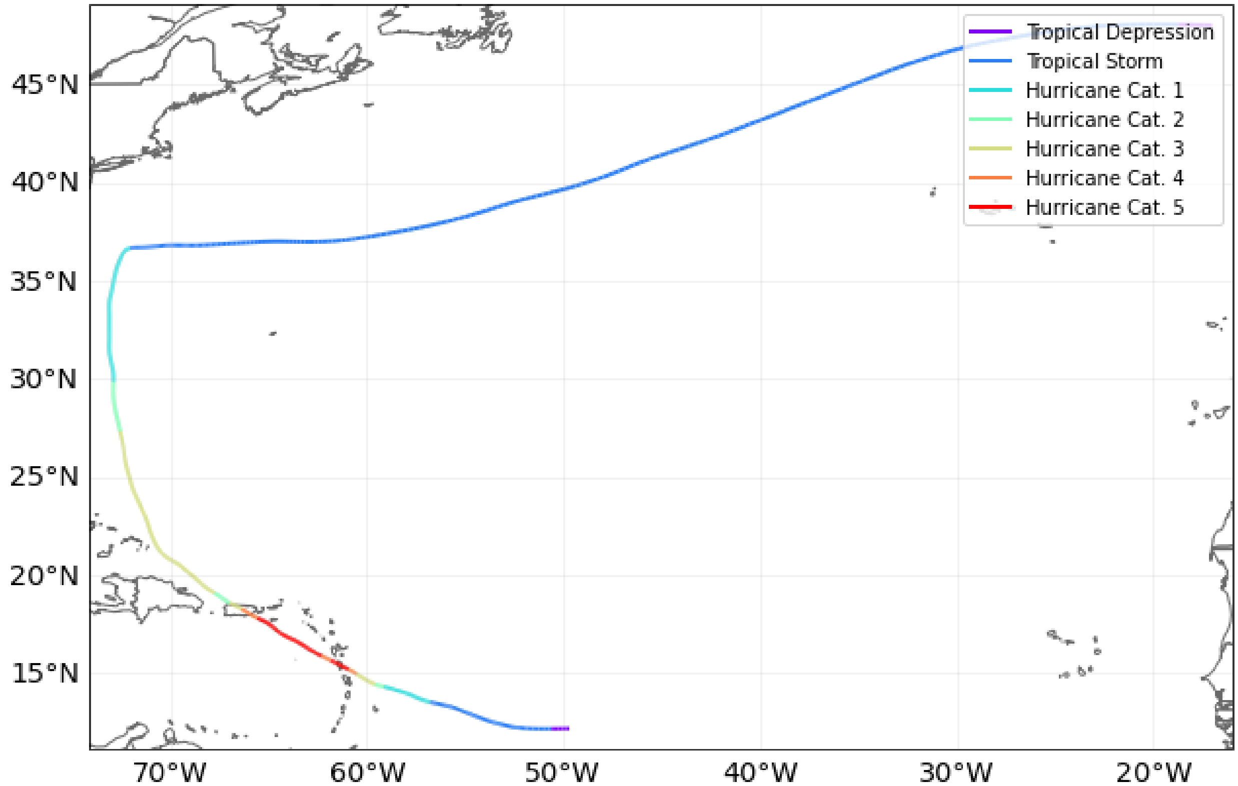
Now we're ready to create our hazard object. This will be a `TropCyclone` class, which inherits from the `Hazard` class, and has the `from_tracks` constructor method to create a hazard from a `TCTracks` object at given centroids.

```
from climada.hazard import TropCyclone

haz = TropCyclone.from_tracks(tracks, centroids=cent)
haz.check()
```

```
2022-03-21 14:35:51,458 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2022-03-21 14:35:52,496 - climada.util.coordinates - INFO - dist_to_coast: UTM 32619␣
↪(1/2)
2022-03-21 14:35:53,234 - climada.util.coordinates - INFO - dist_to_coast: UTM 32620␣
↪(2/2)
2022-03-21 14:35:53,706 - climada.hazard.trop_cyclone - INFO - Mapping 1049 tracks to␣
↪1891 coastal centroids.
2022-03-21 14:35:56,704 - climada.hazard.trop_cyclone - INFO - Progress: 10%
2022-03-21 14:36:00,561 - climada.hazard.trop_cyclone - INFO - Progress: 20%
2022-03-21 14:36:05,356 - climada.hazard.trop_cyclone - INFO - Progress: 30%
2022-03-21 14:36:09,524 - climada.hazard.trop_cyclone - INFO - Progress: 40%
2022-03-21 14:36:15,423 - climada.hazard.trop_cyclone - INFO - Progress: 50%
2022-03-21 14:36:20,307 - climada.hazard.trop_cyclone - INFO - Progress: 60%
2022-03-21 14:36:25,005 - climada.hazard.trop_cyclone - INFO - Progress: 70%
2022-03-21 14:36:30,606 - climada.hazard.trop_cyclone - INFO - Progress: 80%
2022-03-21 14:36:35,743 - climada.hazard.trop_cyclone - INFO - Progress: 90%
2022-03-21 14:36:41,322 - climada.hazard.trop_cyclone - INFO - Progress: 100%
```

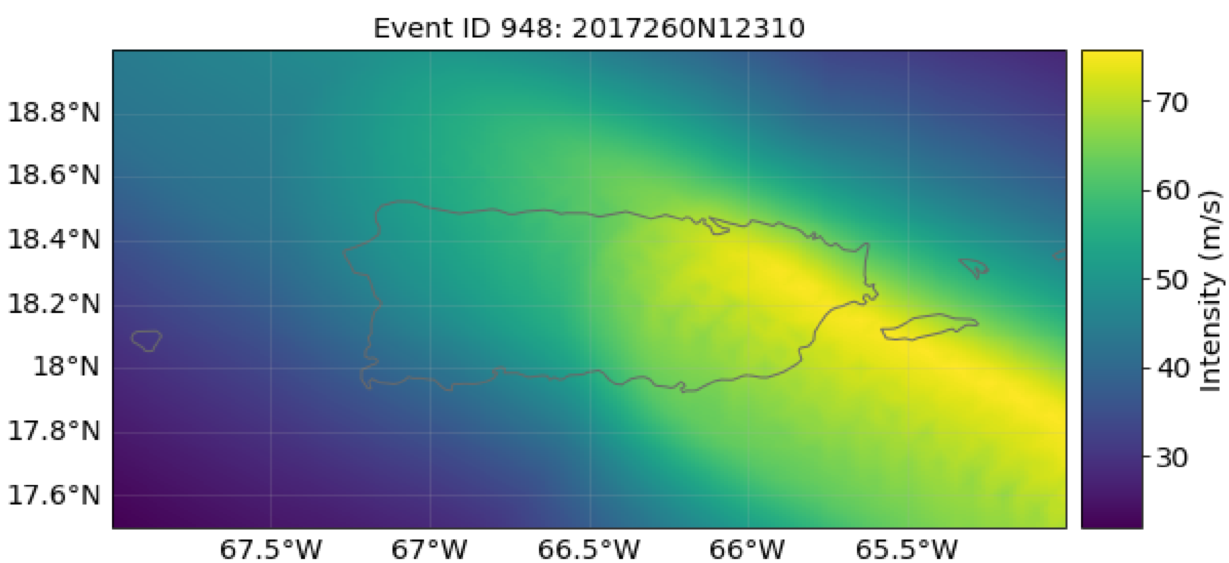
In 2017 Hurricane Maria devastated Puerto Rico. In the IBTRaCs event set, it has ID `2017260N12310` (we use this rather than the name, as IBTRaCS contains three North Atlantic storms called Maria). We can plot the track:

```
tracks.subset({"sid": "2017260N12310"}).plot();  # This is how we subset a TCTracks␣
↪object
```



And plot the hazard on our centroids for Puerto Rico:

```
haz.plot_intensity(event='2017260N12310');
```

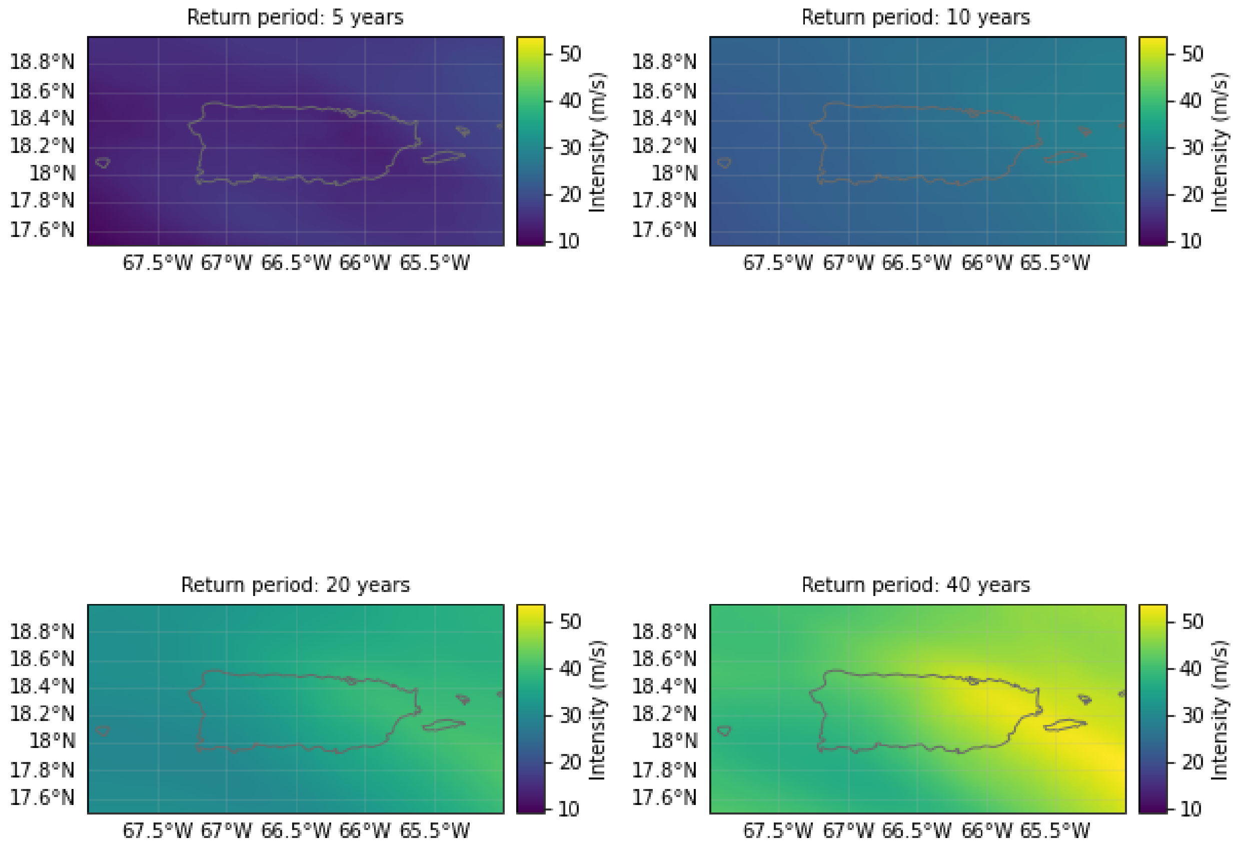


A Hazard object also lets us plot the hazard at different return periods. The IBTRaCS archive produces footprints from

1980 onwards (CLIMADA discarded earlier events) and so the historical period is short. Therefore these plots don't make sense as 'real' return periods, but we're being irresponsible and demonstrating the functionality anyway.

```
haz.plot_rp_intensity(return_periods=(5,10,20,40));
```

```
2022-03-15 22:20:11,511 - climada.hazard.base - INFO - Computing exceedance intenstiy␣
↪map for return periods: [ 5 10 20 40]
```



See the *TropCyclone tutorial* for full details of the TropCyclone hazard class.

We can also recalculate event sets to reflect the effects of climate change. The `apply_climate_scenario_knu` method applies changes in intensity and frequency projected due to climate change, as described in 'Global projections of intense tropical cyclone activity for the late twenty-first century from dynamical downscaling of CMIP5/RCP4.5 scenarios' (Knutson *et al.* 2015). See the *tutorial* for details.

> **Exercise:** Extend this notebook's analysis to examine the effects of climate change in Puerto Rico. You'll need to extend the historical event set with stochastic tracks to create a robust statistical storm climatology - the `TCTracks` class has the functionality to do this. Then you can apply the `apply_climate_scenario_knu` method to the generated hazard object to create a second hazard climatology representing storm activity under climate change. See how the results change using the different hazard sets.

Next we'll work on exposure and vulnerability, part of the Entity class.

6.6 Entity

The entity class is a container class that stores exposures and impact functions (vulnerability curves) needed for a risk calculation, and the discount rates and adaptation measures for an adaptation cost-benefit analysis.

As with Hazard objects, Entities can be read from files or created through code. The Excel template can be found in `climada_python/climada/data/system/entity_template.xlsx`.

In this tutorial we will create an Exposure object using the LitPop economic exposure module, and load a pre-defined wind damage function.

6.6.1 Exposures

The `Entity`'s `exposures` attribute contains geolocalized values of anything exposed to the hazard, whether monetary values of assets or number of human lives, for example. It is of type `Exposures`.

See the *Exposures tutorial* for more detail on the structure of the class, and how to create and import exposures. The *LitPop tutorial* explains how CLIMADA models economic exposures using night-time light and economic data, and is what we'll use here. To combine your exposure with OpenStreetMap's data see the OSM tutorial.

LitPop is a module that allows CLIMADA to estimate exposed populations and economic assets at any point on the planet without additional information, and in a globally consistent way. Before we try it out with the next code block, we'll need to download a data set and put it into the right folder:

1. Go to the download page on Socioeconomic Data and Applications Center (sedac).
2. You'll be asked to log in or register. Please register if you don't have an account.
3. Wait until several drop-down menus show up.
4. Choose in the drop-down menus: Temporal: single year, FileFormat: GeoTiff, Resolution: 30 seconds. Click "2020" and then "create download".
5. Copy the file "gpw_v4_population_count_rev11_2020_30_sec.tif" into the folder "~/climada/data". (Or you can run the block once to find the right path in the error message)

Now we can create an economic Exposure dataset for Puerto Rico.

```
from climada.entity.exposures import LitPop

exp_litpop = LitPop.from_countries('Puerto Rico', res_arcsec = 120)  # We'll go lower␣
↪resolution than default to keep it simple
exp_litpop.set_geometry_points() # Set geodataframe geometries from lat lon data

exp_litpop.plot_hexbin(pop_name=True, linewidth=4, buffer=0.1);
```

```
2022-03-21 14:37:03,770 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: PRI (630)...

2022-03-21 14:37:03,773 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2022-03-21 14:37:03,774 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-03-21 14:37:03,824 - climada.entity.exposures.litpop.nightlight - INFO - No␣
↪satellite files found locally in /home/yuyue/climada/data
2022-03-21 14:37:03,826 - climada.entity.exposures.litpop.nightlight - INFO -␣
↪Attempting to download file from https://eoimages.gsfc.nasa.gov/images/imagerecords/
↪144000/144897/BlackMarble_2016_B1_geo_gray.tif
```

(continues on next page)

(continued from previous page)

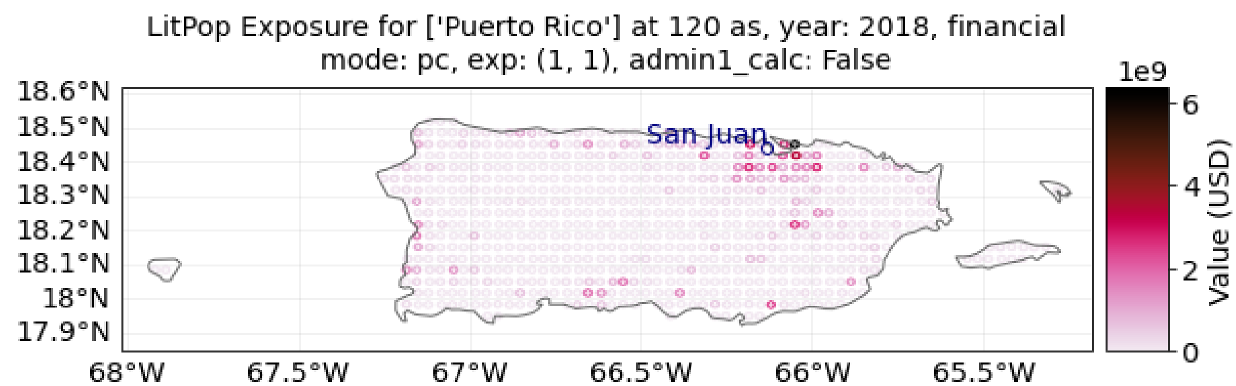
```
2022-03-21 14:37:04,665 - climada.util.files_handler - INFO - Downloading https://
↪eoimages.gsfc.nasa.gov/images/imagerecords/144000/144897/BlackMarble_2016_B1_geo_
↪gray.tif to file /home/yuyue/climada/data/BlackMarble_2016_B1_geo_gray.tif
```

```
26.8kKB [00:02, 9.72kKB/s]
```

```
2022-03-21 14:37:08,919 - climada.util.files_handler - INFO - Downloading https://
↪databank.worldbank.org/data/download/Wealth-Accounts_CSV.zip to file /mnt/c/Users/
↪yyljy/Documents/climada_main/doc/tutorial/results/Wealth-Accounts_CSV.zip
```

```
1.44kKB [00:03, 429KB/s]
```

```
2022-03-21 14:37:12,440 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-03-21 14:37:13,356 - climada.util.finance - INFO - GDP PRI 2018: 1.009e+11.
2022-03-21 14:37:13,361 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-03-21 14:37:13,378 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-03-21 14:37:13,380 - climada.entity.exposures.base - INFO - category_id not set.
2022-03-21 14:37:13,381 - climada.entity.exposures.base - INFO - cover not set.
2022-03-21 14:37:13,383 - climada.entity.exposures.base - INFO - deductible not set.
2022-03-21 14:37:13,384 - climada.entity.exposures.base - INFO - centr_ not set.
2022-03-21 14:37:13,387 - climada.util.coordinates - INFO - Setting geometry points.
```



LitPop's default exposure is measured in US Dollars, with a reference year depending on the most recent data available.

Once we've created our impact function we will come back to this Exposure and give it the parameters needed to connect exposure to impacts.

6.6.2 Impact functions

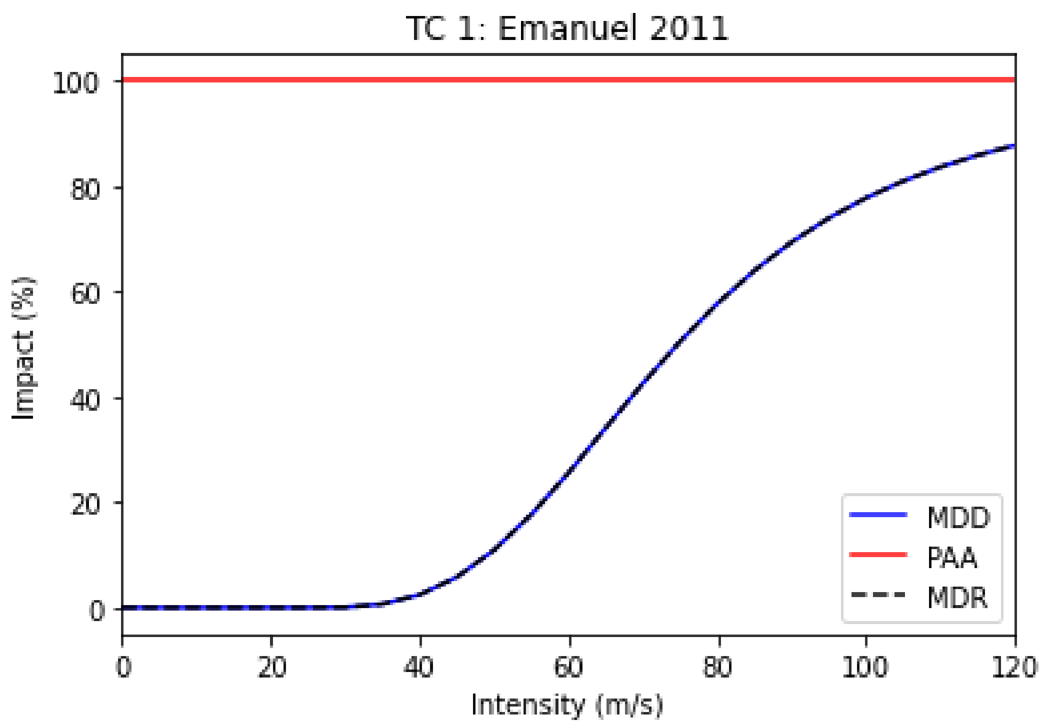
Impact functions describe a relationship between a hazard's intensity and your exposure in terms of a percentage loss. The impact is described through two terms. The Mean Degree of Damage (MDD) gives the percentage of an exposed asset's numerical value that's affected as a function of intensity, such as the damage to a building from wind in terms of its total worth. Then the Proportion of Assets Affected (PAA) gives the fraction of exposures that are affected, such as the mortality rate in a population from a heatwave. These multiply to give the Mean Damage Ratio (MDR), the average impact to an asset.

Impact functions are stored as the Entity's `impact_funcs` attribute, in an instance of the `ImpactFuncSet` class which groups one or more `ImpactFunc` objects. They can be specified manually, read from a file, or you can use CLIMADA's pre-defined impact functions. We'll use a pre-defined function for tropical storm wind damage stored in the `IFTropCyclone` class.

See the *Impact Functions tutorial* for a full guide to the class, including how data are stored and reading and writing to files.

We initialise an Impact Function with the `IFTropCyclone` class, and use its `from_emanuel_usa` method to load the Emanuel (2011) impact function. (The class also contains regional impact functions for the full globe, but we'll won't use these for now.) The class's `plot` method visualises the function, which we can see is expressed just through the Mean Degree of Damage, with all assets affected.

```
from climada.entity.impact_funcs import ImpactFuncSet, ImpfTropCyclone

imp_fun = ImpfTropCyclone.from_emanuel_usa()
imp_fun.plot();
```



The plot title also includes information about the function's ID, which were also set by the `from_emanuel_usa` class method. The hazard is "TC" and the function ID is 1. Since a study might use several impact functions - for different hazards, or for different types of exposure.

We then create an `ImpactFuncSet` object to store the impact function. This is a container class, and groups a study's impact functions together. Studies will often have several impact functions, due to multiple hazards, multiple types of exposure that are impacted differently, or different adaptation scenarios. We add it to our Entity object.

```
imp_fun_set = ImpactFuncSet([imp_fun])
```

Finally, we can update our LitPop exposure to point to the TC 1 impact function. This is done by adding a column to the exposure:

```
exp_litpop.gdf['impf_TC'] = 1
```

```
2022-03-21 14:37:53,587 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-03-21 14:37:53,591 - climada.entity.exposures.base - INFO - category_id not set.
2022-03-21 14:37:53,593 - climada.entity.exposures.base - INFO - cover not set.
2022-03-21 14:37:53,594 - climada.entity.exposures.base - INFO - deductible not set.
2022-03-21 14:37:53,595 - climada.entity.exposures.base - INFO - centr_ not set.
2022-03-21 14:37:53,600 - climada.entity.impact_funcs.base - WARNING - For intensity␣
↪= 0, mdd != 0 or paa != 0. Consider shifting the origin of the intensity scale. In␣
↪impact.calc the impact is always null at intensity = 0.
```

Here the `impf_TC` column tells the CLIMADA engine that for a tropical cyclone (TC) hazard, it should use the first impact function defined for TCs. We use the same impact function for all of our exposure.

This is now everything we need for a risk analysis, but while we're working on the Entity class, we can define the adaptation measures and discount rates needed for an adaptation analysis. If you're not interested in the cost-benefit analysis, you can skip ahead to the *Impact section*

6.6.3 Adaptation measures

CLIMADA's adaptation measures describe possible interventions that would change event hazards and impacts, and the cost of these interventions.

They are stored as `Measure` objects within a `MeasureSet` container class (similarly to `ImpactFuncSet` containing several `ImpactFuncs`), and are assigned to the `measures` attribute of the Entity.

See the *Adaptation Measures tutorial* on how to create, read and write measures. CLIMADA doesn't yet have pre-defined adaptation measures, mostly because they are hard to standardise.

The best way to understand an adaptation measure is by an example. Here's a possible measure for the creation of coastal mangroves (ignore the exact numbers, they are just for illustration):

```python
from climada.entity import Measure, MeasureSet

meas_mangrove = Measure(
    name='Mangrove',
    haz_type='TC',
    color_rgb=np.array([0.2, 0.2, 0.7]),
    cost=500000000,
    mdd_impact=(1, 0),
    paa_impact=(1, -0.15),
    hazard_inten_imp=(1, -10),
)

meas_set = MeasureSet(measure_list=[meas_mangrove])
meas_set.check()
```

What values have we set here?

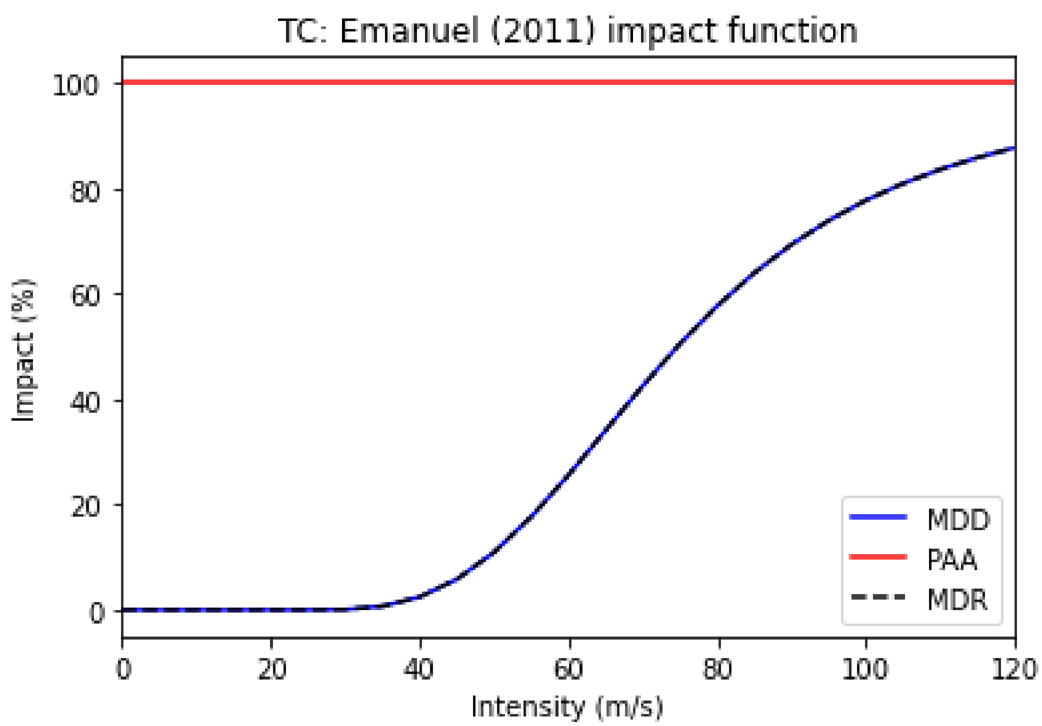
- The `haz_type` gives the hazard that this measure affects.
- The `cost` is a flat price that will be used in cost-benefit analyses.
- The `mdd_impact`, `paa_impact`, and `hazard_inten_imp` attributes are all tuples that describes a linear transformation to event hazard, the impact function's mean damage degree and the impact function's proportion of assets affected. The tuple `(a, b)` describes a scalar multiplication of the function and a constant to add. So `(1, 0)` is unchanged, `(1.1, 0)` increases values by 10%, and `(1, -10)` decreases all values by 10.

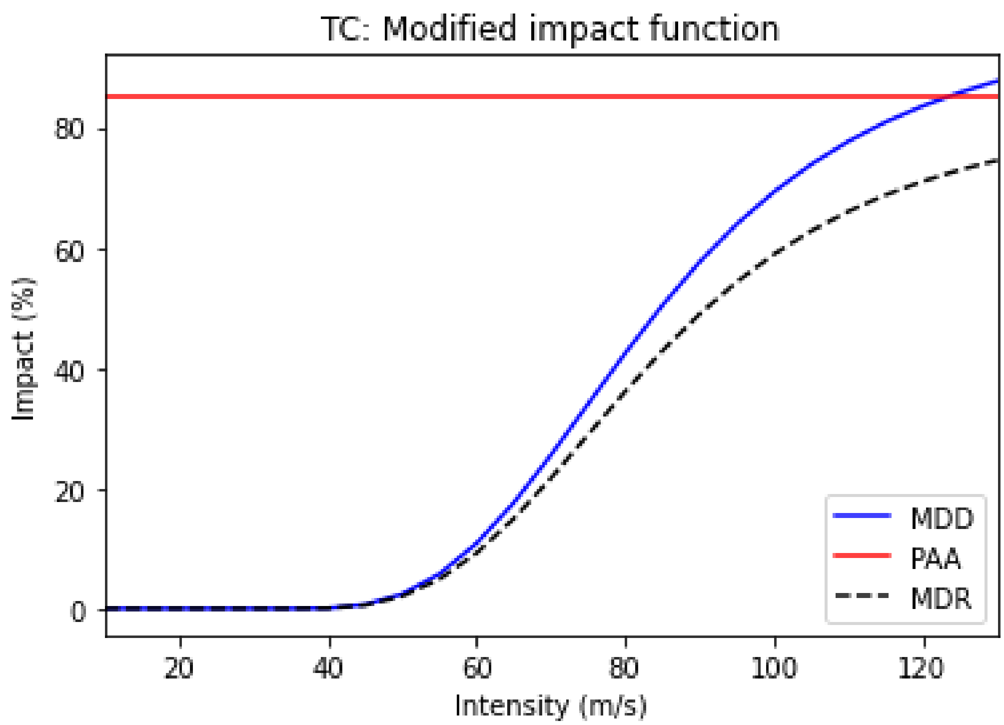
So the Mangrove example above costs 50,000,000 USD, protects 15% of assets from any impact at all (`paa_impact = (1, -0.15)`) and decreases the (effective) hazard intensity by 10 m/s (`hazard_inten_imp = (1, -10)`.

We can apply these measures to our existing Exposure, Hazard and Impact functions, and plot the old and new impact functions:

```
mangrove_exp, mangrove_imp_fun_set, mangrove_haz = meas_mangrove.apply(exp_litpop,␣
↪imp_fun_set, haz)
axes1 = imp_fun_set.plot()
axes1.set_title('TC: Emanuel (2011) impact function')
axes2 = mangrove_imp_fun_set.plot()
axes2.set_title('TC: Modified impact function')
```

```
Text(0.5, 1.0, 'TC: Modified impact function')
```





Let's define a second measure. Again, the numbers here are made up, for illustration only.

```
meas_buildings = Measure(
    name='Building code',
    haz_type='TC',
    color_rgb=np.array([0.2, 0.7, 0.5]),
    cost=100000000,
    hazard_freq_cutoff=0.1,
)

meas_set.append(meas_buildings)
meas_set.check()

buildings_exp, buildings_imp_fun_set, buildings_haz = meas_buildings.apply(exp_litpop,
↪ imp_fun_set, haz)
```

```
2022-03-21 14:38:24,711 - climada.entity.exposures.base - INFO - Matching 691␣
↪exposures with 1891 centroids.
2022-03-21 14:38:24,716 - climada.engine.impact - INFO - Calculating damage for 661␣
↪assets (>0) and 1049 events.
```

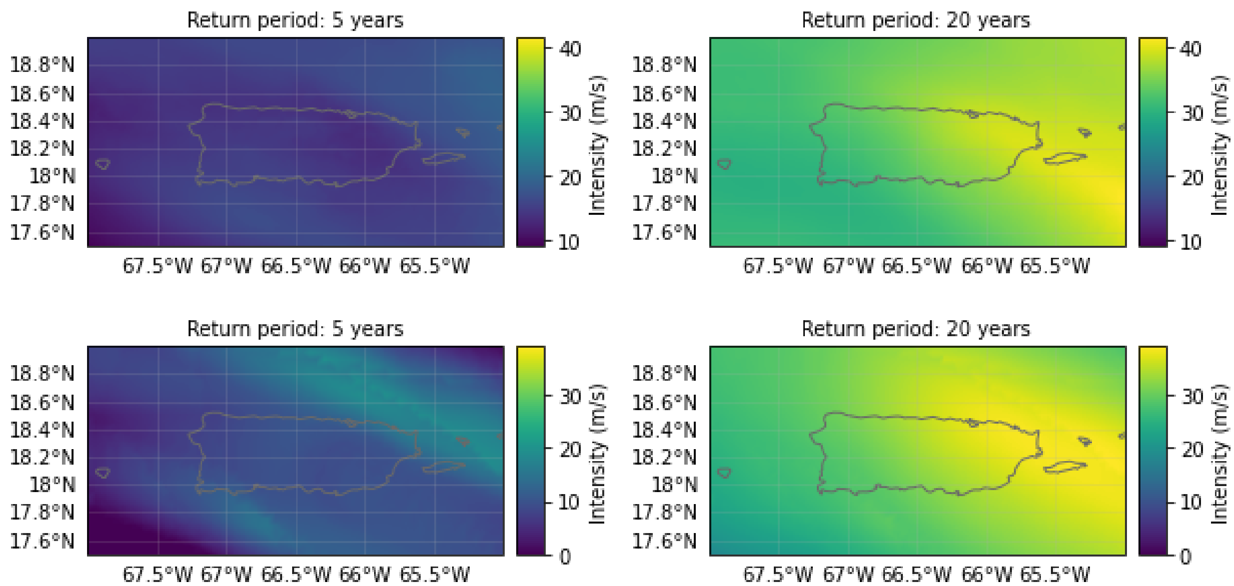
This measure describes an upgrade to building codes to withstand 10-year events. The measure costs 100,000,000 USD and, through `hazard_freq_cutoff = 0.1`, removes events with calculated impacts below the 10-year return period.

The *Adaptation Measures tutorial* describes other parameters for describing adaptation measures, including risk transfer, assigning measures to subsets of exposure, and reassigning impact functions.

We can compare the 5- and 20-year return period hazard (remember: not a real return period due to the small event set!) compared to the adjusted hazard once low-impact events are removed.

```
haz.plot_rp_intensity(return_periods=(5, 20));
buildings_haz.plot_rp_intensity(return_periods=(5, 20));
```

```
2022-03-15 22:27:56,309 - climada.hazard.base - INFO - Computing exceedance intenstiy␣
↪map for return periods: [ 5 20]
2022-03-15 22:28:13,337 - climada.hazard.base - INFO - Computing exceedance intenstiy␣
↪map for return periods: [ 5 20]
2022-03-15 22:28:13,911 - climada.hazard.base - WARNING - Exceedance intenstiy values␣
↪below 0 are set to 0.                  Reason: no negative intensity values were␣
↪found in hazard.
```



It shows there are now very few events at the 5-year return period - the new building codes removed most of these from the event set.

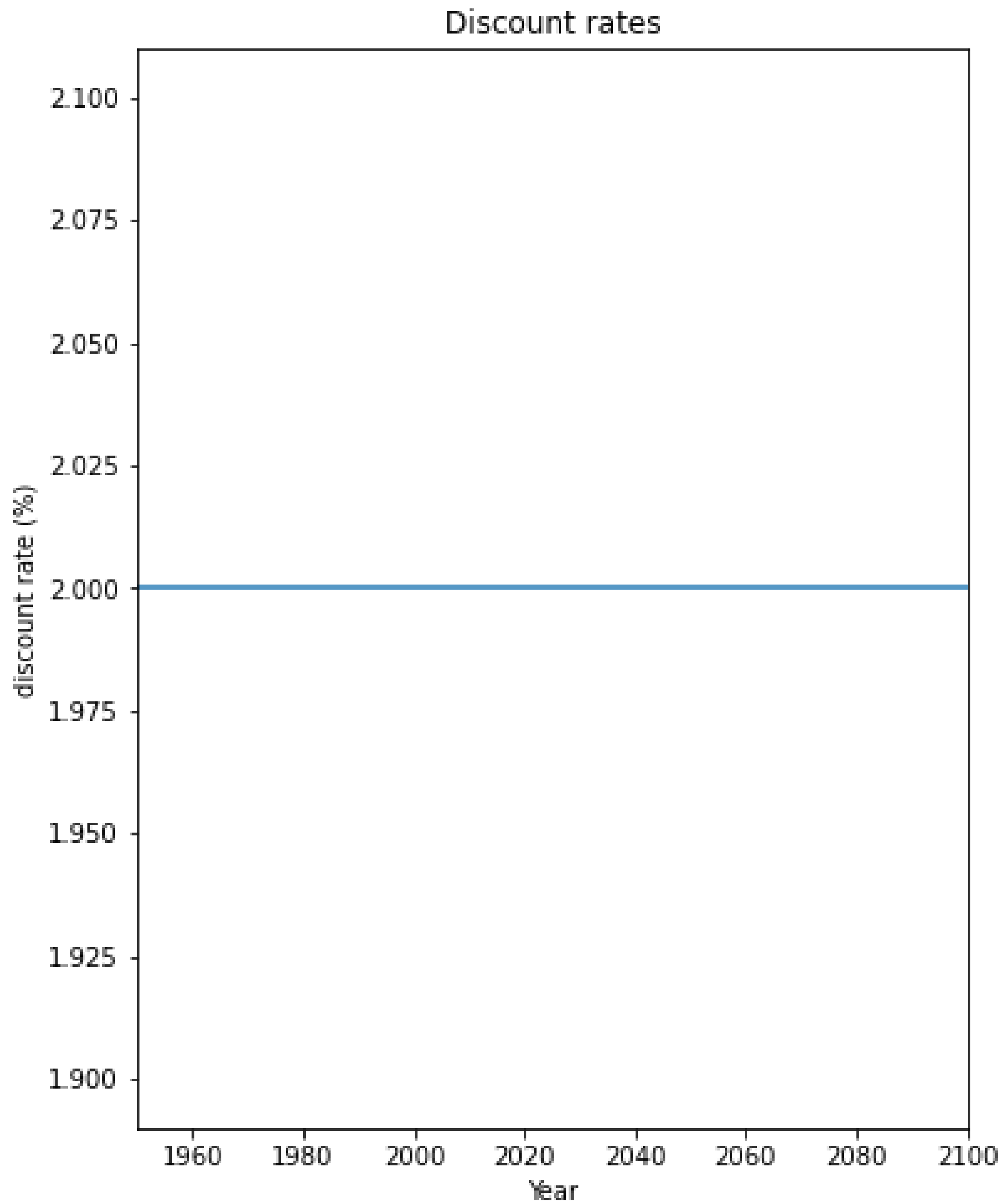
6.6.4 Discount rates

The `disc_rates` attribute is of type `DiscRates`. This class contains the discount rates for the following years and computes the net present value for given values.

See the *Discount Rates tutorial* for more details about creating, reading and writing the `DiscRates` class, and how it is used in calculations.

Here we will implement a simple, flat 2% discount rate.

```
from climada.entity import DiscRates

years=np.arange(1950, 2101)
rates=np.ones(years.size) * 0.02
disc = DiscRates(years=years, rates=rates)
disc.check()
disc.plot()
```



We are now ready to move to the last part of the CLIMADA model for Impact and Cost Benefit analyses.

6.6.5 Define Entity

We are now ready to define our Entity object that contains the exposures, impact functions, discount rates and measures.

```
from climada.entity import Entity

ent = Entity(
    exposures=exp_litpop,
    disc_rates=disc,
    impact_func_set=imp_fun_set,
    measure_set=meas_set
)
```

6.7 Engine

The CLIMADA Engine is where the main risk calculations are done. It contains two classes, `Impact`, for risk assessments, and `CostBenefit`, to evaluate adaptation measures.

6.7.1 Impact

Let us compute the impact of historical tropical cyclones in Puerto Rico.

Our work above has given us everything we need for a risk analysis using the Impact class. By computing the impact for each historical event, the `Impact` class provides different risk measures, as the expected annual impact per exposure, the probable maximum impact for different return periods and the total average annual impact.

Note: the configurable parameter `CONFIG.maz_matrix_size` controls the maximum matrix size contained in a chunk. You can decrease its value if you are having memory issues when using the `Impact`'s `calc` method. A high value will make the computation fast, but increase the memory use. (See the *config guide* on how to set configuration values.)

CLIMADA calculates impacts by providing exposures, impact functions and hazard to an `Impact` object's `calc` method:

```
from climada.engine import ImpactCalc

imp = ImpactCalc(ent.exposures, ent.impact_funcs, haz).impact()
```

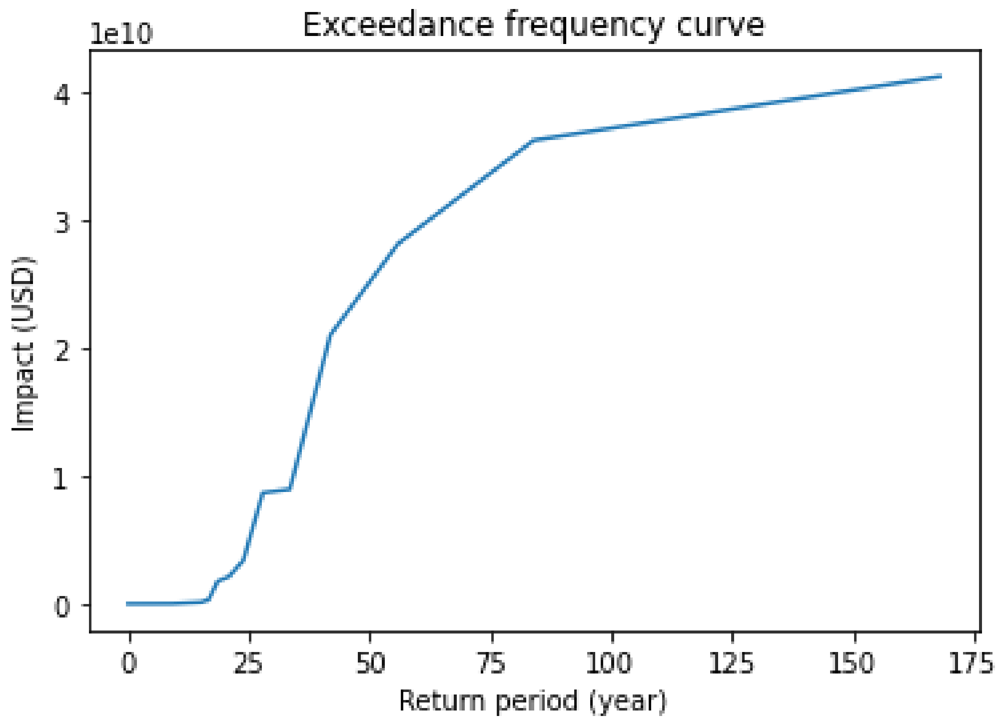
```
2022-03-21 14:38:36,337 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-21 14:38:36,343 - climada.engine.impact - INFO - Calculating damage for 661␣
↪assets (>0) and 1049 events.
```

A useful parameter for the `calc` method is `save_mat`. When set to `True` (default is `False`), the `Impact` object saves the calculated impact for each event at each point of exposure, stored as a (large) sparse matrix in the `imp_mat` attribute. This allows for more detailed analysis at the event level.

The `Impact` class includes a number of analysis tools. We can plot an exceedance frequency curve, showing us how often different damage thresholds are reached in our source data (remember this is only 40 years of storms, so not a full climatology!)

```
freq_curve = imp.calc_freq_curve() # impact exceedance frequency curve
freq_curve.plot();

print('Expected average annual impact: {:.3e} USD'.format(imp.aai_agg))
```

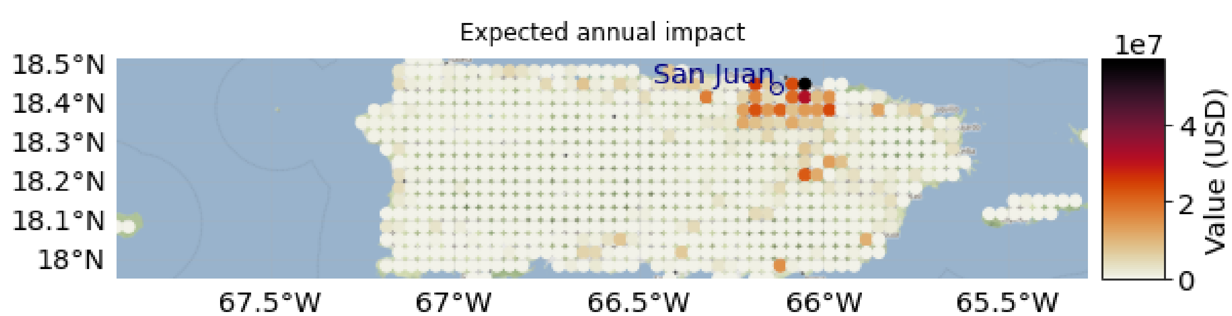
```
Expected average annual impact: 9.068e+08 USD
```



We can map the expected annual impact by exposure:

```
imp.plot_basemap_eai_exposure(buffer=0.1); # average annual impact at each exposure
```

```
2022-03-21 14:38:43,047 - climada.util.coordinates - INFO - Setting geometry points.
2022-03-21 14:38:43,151 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-03-21 14:38:46,480 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```



For additional functionality, including plotting the impacts of individual events, see the *Impact tutorial*.

> **Exercise:** Plot the impacts of Hurricane Maria. To do this you'll need to set `save_mat=True` in the earlier `ImpactCalc.impact()`.

We can save our variables in pickle format using the `save` function and load them with `load`. This will save your results in the folder specified in the configuration file. The default folder is a `results` folder which is created in the current path (see default configuration file `climada/conf/defaults.conf`). However, we recommend to use

CLIMADA's writers in hdf5 or csv whenever possible.

```
import os
from climada.util import save, load

### Uncomment this to save - saves by default to ./results/
# save('impact_puerto_rico_tc.p', imp)

### Uncomment this to read the saved data:
# abs_path = os.path.join(os.getcwd(), 'results/impact_puerto_rico_tc.p')
# data = load(abs_path)
```

Impact also has write_csv() and write_excel() methods to save the impact variables, and write_sparse_csr() to save the impact matrix (impact per event and exposure). Use the *Impact tutorial* to get more information about these functions and the class in general.

6.7.2 Adaptation options appraisal

Finally, let's look at a cost-benefit analysis. The adaptation measures defined with our Entity can be valued by estimating their cost-benefit ratio. This is done in the class CostBenefit.

Let us suppose that the socioeconomic and climatological conditions remain the same in 2040. We then compute the cost and benefit of every adaptation measure from our Hazard and Entity (and plot them) as follows:

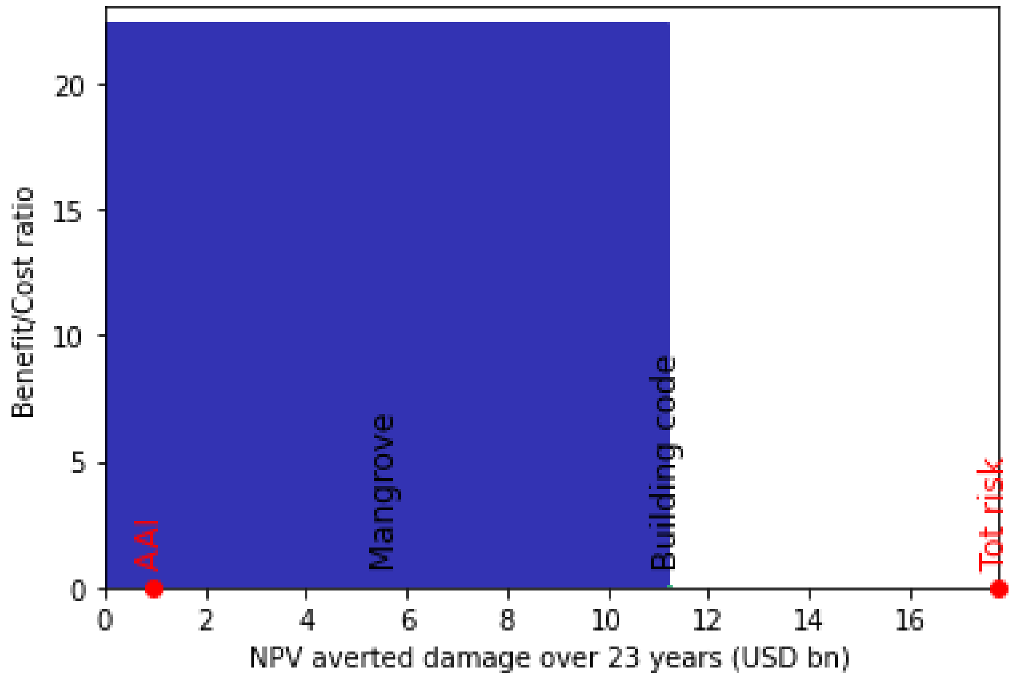
```
from climada.engine import CostBenefit

cost_ben = CostBenefit()
cost_ben.calc(haz, ent, future_year=2040) # prints costs and benefits
cost_ben.plot_cost_benefit(); # plot cost benefit ratio and averted damage of every␣
↪exposure
cost_ben.plot_event_view(return_per=(10, 20, 40)); # plot averted damage of each␣
↪measure for every return period
```

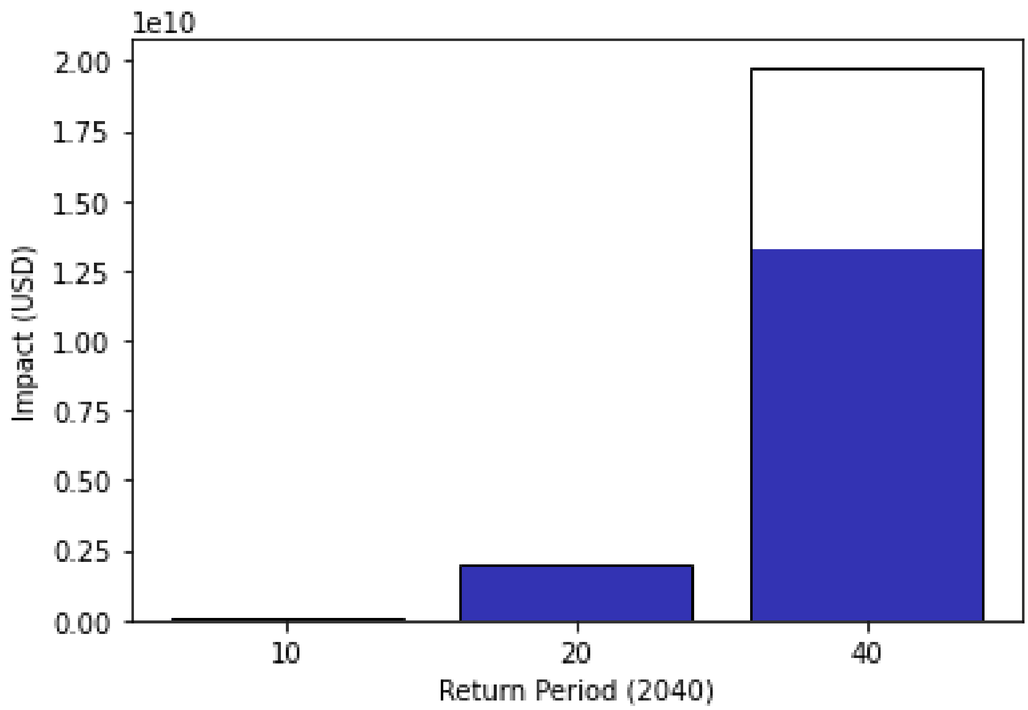
```
2022-03-15 22:32:07,393 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-15 22:32:07,397 - climada.engine.impact - INFO - Calculating damage for 691␣
↪assets (>0) and 1040 events.
2022-03-15 22:32:07,406 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-15 22:32:07,408 - climada.engine.impact - INFO - Calculating damage for 691␣
↪assets (>0) and 1040 events.
2022-03-15 22:32:07,418 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-15 22:32:07,420 - climada.engine.impact - INFO - Calculating damage for 691␣
↪assets (>0) and 1040 events.
2022-03-15 22:32:07,437 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-15 22:32:07,440 - climada.engine.impact - INFO - Calculating damage for 691␣
↪assets (>0) and 1040 events.
2022-03-15 22:32:07,452 - climada.engine.cost_benefit - INFO - Computing cost benefit␣
↪from years 2018 to 2040.

Measure          Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-------------  ---------------  ------------------  --------------
Mangrove                   0.5          11.2129          22.4258
Building code              0.1           0.00761204       0.0761204
```

(continues on next page)

(continued from previous page)

```
--------------------  ---------  --------
Total climate risk:   17.749     (USD bn)
Average annual risk:   0.951281  (USD bn)
Residual risk:         6.52855   (USD bn)
--------------------  ---------  --------
Net Present Values
```





This is just the start. Analyses improve as we add more adaptation measures into the mix.

Cost-benefit calculations can also include

- climate change, by specifying the `haz_future` parameter in `CostBenefit.calc()`
- changes to economic exposure over time (or to whatever exposure you're modelling) by specifying the `ent_future` parameter in `CostBenefit.calc()`
- different functions to calculate risk benefits. These are specified in `CostBenefit.calc()` and by default use changes to average annual impact
- linear, sublinear and superlinear evolution of impacts between the present and future, specified in the `imp_time_depen` parameter in `CostBenefit.calc()`

And once future hazards and exposures are defined, we can express changes to impacts over time as waterfall diagrams. See the CostBenefit class for more details.

> **Exercise:** repeat the above analysis, creating future climate hazards (see the first exercise), and future exposures based on projected economic growth. Visualise it with the `CostBenefit.plot_waterfall()` method.

6.8 What next?

Thanks for following this tutorial! Take time to work on the exercises it suggested, or design your own risk analysis for your own topic. More detailed tutorials for individual classes were listed in the *Features* section.

Also, explore the full CLIMADA documentation and additional resources *described at the start of this document* to learn more about CLIMADA, its structure, its existing applications and how you can contribute.

CHAPTER SEVEN

FAST AND BASIC PYTHON INTRODUCTION

Prepared by G. Aznar Siguan

Most of the examples come from the official Python tutorial: https://docs.python.org/3/tutorial/

7.1 Contents

- *Numbers and Strings*
- *Lists*
- *Tuples*
- *Sets*
- *Dictonaries*
- *Functions*
- *Objects*

7.2 Numbers and Strings

The operators +, -, * and / work just like in most other languages:

```
print('Addition: 2 + 2 =', 2 + 2)
print('Substraction: 50 - 5*6 =', 50 - 5*6)
print('Use of parenthesis: (50 - 5*6) / 4 =', (50 - 5*6) / 4)
print('Classic division returns a float: 17 / 3 =', 17 / 3)
print('Floor division discards the fractional part: 17 // 3 =', 17 // 3)
print('The % operator returns the remainder of the division: 17 % 3 =', 17 % 3)
print('Result * divisor + remainder: 5 * 3 + 2 =', 5 * 3 + 2)
print('5 squared: 5 ** 2 =', 5 ** 2)
print('2 to the power of 7: 2 ** 7 =', 2 ** 7)
```

The integer numbers (e.g. 2, 4, 20) have type int, the ones with a fractional part (e.g. 5.0, 1.6) have type float. Operators with mixed type operands convert the integer operand to floating point:

```
tax = 12.5 / 100
price = 100.50
price * tax
```

Strings can be enclosed in single quotes ('…') or double quotes ("…") with the same result. \ can be used to escape quotes.

If you don't want characters prefaced by \ to be interpreted as special characters, you can use raw strings by adding an r before the first quote.

```
print('spam eggs')  # single quotes
print('doesn\'t')   # use \' to escape the single quote...
print("doesn't")    # ...or use double quotes instead
print('"Yes," he said.')
print("\"Yes,\" he said.")
print('"Isn\'t," she said.')
```

Strings can be indexed (subscripted), with the first character having index 0.

Indices may also be negative numbers, to start counting from the right. Note that since -0 is the same as 0, negative indices start from -1.

```
word = 'Python'
print('word = ', word)
print('Character in position 0: word[0] =', word[0])
print('Character in position 5: word[5] =', word[5])
print('Last character: word[-1] =', word[-1])
print('Second-last character: word[-2] =', word[-2])
print('word[-6] =', word[-6])
```

In addition to indexing, slicing is also supported. While indexing is used to obtain individual characters, slicing allows you to obtain substring:

```
print('Characters from position 0 (included) to 2 (excluded): word[0:2] =', word[0:2])
print('Characters from position 2 (included) to 5 (excluded): word[2:5] =', word[2:5])
```

7.3 Lists

Lists can be written as a list of comma-separated values (items) between square brackets. Lists might contain items of different types, but usually the items all have the same type.

Like strings (and all other built-in sequence type), lists can be indexed and sliced:

```
squares = [1, 4, 9, 16, 25]
print('squares: ', squares)
print('Indexing returns the item: squares[0]:', squares[0])
print('squares[-1]:', squares[-1])
print('Slicing returns a new list: squares[-3:]:', squares[-3:])
print('squares[:]:', squares[:])
```

Lists also support operations like concatenation:

```
squares + [36, 49, 64, 81, 100]
```

Unlike strings, which are immutable, lists are a mutable type, i.e. it is possible to change their content:

```
cubes = [1, 8, 27, 65, 125]  # something's wrong here
cubes[3] = 64  # replace the wrong value
cubes.append(216)  # add the cube of 6
cubes.append(7 ** 3)  # and the cube of 7
cubes
```

```
# Note: execution of this cell will fail

# Try to modify a character of a string
word = 'Python'
word[0] = 'p'
```

List comprehensions provide a concise way to create lists. Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.

```
squares = []
for x in range(10):
    squares.append(x**2)
squares
```

```
# lambda functions: functions that are not bound to a name, e.g lambda x: x**2
# Map applies a function to all the items in an input_list: map(function_to_apply,␣
↪list_of_inputs)
squares = list(map(lambda x: x**2, range(10)))
squares
```

```
squares = [x**2 for x in range(10)]
squares
```

7.4 Tuples

A tuple consists of a number of values separated by commas, for instance:

```
t = 12345, 54321, 'hello!'
t[0]
```

```
t
```

```
# Tuples may be nested:
u = t, (1, 2, 3, 4, 5)
u
```

```
# Note: execution of this cell will fail

# Tuples are immutable:
t[0] = 88888
```

```
# but they can contain mutable objects:
v = ([1, 2, 3], [3, 2, 1])
v
```

Tuples are immutable, and usually contain a heterogeneous sequence of elements that are accessed via unpacking or indexing. Lists are mutable, and their elements are usually homogeneous and are accessed by iterating over the list.

```
t = 12345, 54321, 'hello!' # tuple packing
x, y, z = t # tuple unpacking
x, y, z
```

7.5 Sets

A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the set() function can be used to create sets.

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
basket # show that duplicates have been removed
```

```
'orange' in basket # fast membership testing
```

```
'crabgrass' in basket
```

```
# Demonstrate set operations on unique letters from two words
a = set('abracadabra')
b = set('alacazam')
a # unique letters in a
```

```
a - b # letters in a but not in b
```

```
a | b # letters in a or b or both
```

```
a & b # letters in both a and b
```

```
a ^ b # letters in a or b but not both
```

```
# check set documentation
help(set)
```

```
# check which methods can be applied on set
dir(set)
```

```
# Define a new set and try some set methods (freestyle)
```

7.6 Dictionaries

Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys.

It is best to think of a dictionary as an unordered set of key: value pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}. Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary; this is also the way dictionaries are written on output.

```
tel = {'jack': 4098, 'sape': 4139}
tel['guido'] = 4127
tel
```

```
tel['jack']
```

```
del tel['sape']
```

```
tel['irv'] = 4127
tel
```

```
list(tel.keys())
```

```
sorted(tel.keys())
```

```
'guido' in tel
```

```
'jack' not in tel
```

7.7 Functions

We can create a function that writes the Fibonacci series to an arbitrary boundary:

```
def fib(n):    # write Fibonacci series up to n
     """Print a Fibonacci series up to n."""
     a, b = 0, 1             # two assignments in one line
     while a < n:
         print(a, end=' ')
         a, b = b, a+b       # two assignments in one line
     print()
```

```
# Now call the function we just defined:
fib(2000)
```

The value of the function name has a type that is recognized by the interpreter as a user-defined function. This value can be assigned to another name which can then also be used as a function. This serves as a general renaming mechanism:

```
print(fib)
print(type(fib)) # function type
f = fib
f(100)
```

Be careful when using mutable types as inputs in functions, as they might be modified:

```
def dummy(x):
    x += x

xx = 5
print('xx before function call: ', xx)
dummy(xx)
print('xx after function call: ', xx)

yy = [5]
print('yy before function call: ', yy)
dummy(yy)
print('yy after function call: ', yy)
```

7.7.1 Default argument values:

The most useful form is to specify a default value for one or more arguments. This creates a function that can be called with fewer arguments than it is defined to allow. For example:

```
def ask_ok(prompt, retries=4, reminder='Please try again!'):
    while True:
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise ValueError('invalid user response')
        print(reminder)
```

```
#This function can be called in several ways:

#giving only the mandatory argument:
ask_ok('Do you really want to quit?')
```

```
# giving one of the optional arguments:
ask_ok('OK to overwrite the file?', 2)
```

```
# or even giving all arguments:
ask_ok('OK to overwrite the file?', 2, 'Come on, only yes or no!')
```

Functions can also be called using keyword arguments of the form kwarg=value:

```
ask_ok('OK to overwrite the file?', reminder='Come on, only yes or no!')
```

Default `None` values: `None` default values can be used to handle optional parameters.

```
def test(x=None):
    if x is None:
        print('no x here')
    else:
        print(x)
test()
```

7.8 Objects

Example class definition:

```
class Dog:                    # same as "class Dog(object)"

    kind = 'canine'           # class variable shared by all instances

    def __init__(self, name): # initialization method
        self.name = name      # instance variable unique to each instance
        self.tricks = []      # creates a new empty list for each dog

```

(continues on next page)

(continued from previous page)

```
    def add_trick(self, trick):    # class method
        self.tricks.append(trick)
```

When a class defines an *_init_*() method, class instantiation automatically invokes *_init_*() for the newly-created class instance:

```
d = Dog('Fido') # creates a new instance of the class and assigns this object to the␣
↪local variable d
d.name
```

```
e = Dog('Buddy') # creates a new instance of the class and assigns this object to the␣
↪local variable e
d.add_trick('roll over')
e.add_trick('play dead')
```

```
d.tricks # unique to d
```

```
e.tricks # unique to e
```

```
d.kind # shared by all dogs
```

```
e.kind # shared by all dogs
```

7.8.1 Inheritance:

The syntax for a derived class definition looks like this:

`class DerivedClassName(BaseClassName)`

A derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. Example:

```
class Animal:          # base class

    def __init__(self, kind):
        self.kind = kind
        self.tricks = []

    def add_trick(self, trick):    # class method
        self.tricks.append(trick)

class Dog(Animal):    # derived class

    def __init__(self): # override of __init__ base method
        super(Dog, self).__init__('canine') # call Animal __init__ method with input␣
↪string
```

```
fido = Dog() # fido is automatically an animal of kind 'canine'
print(fido.kind)
fido.add_trick('play dead') # Dog class can use Animal class
print(fido.tricks)
```

Python supports a form of multiple inheritance as well. A class definition with multiple base classes looks like this:

```
class DerivedClassName(Base1, Base2, Base3):
```

7.8.2 Private Variables and Methods:

"Private" instance variables that cannot be accessed except from inside an object don't exist in Python. However, there is a convention that is followed by most Python code: a name prefixed with an underscore (e.g. _spam) should be treated as a non-public part of the API (whether it is a function, a method or a data member).

Example of internal class use of private method __update. The user is not meant to use __update, but update. However, *_update can be used internally to be called from the* _init method:

```python
class Mapping:
    def __init__(self, iterable):
        self.items_list = []
        self.__update(iterable)

    def update(self, iterable):
        for item in iterable:
            self.items_list.append(item)

    __update = update   # private copy of original update() method

class MappingSubclass(Mapping):

    def update(self, keys, values):
        # provides new signature for update()
        # but does not break __init__()
        for item in zip(keys, values):
            self.items_list.append(item)
```

CHAPTER

EIGHT

HAZARD TUTORIALS

8.1 Hazard class

8.1.1 What is a hazard?

A hazard describes weather events such as storms, floods, droughts, or heat waves both in terms of probability of occurrence as well as physical intensity.

8.1.2 How are hazards embedded in the CLIMADA architecture?

Hazards are defined by the base class `Hazard` which gathers the required attributes that enable the impact computation (such as centroids, frequency per event, and intensity per event and centroid) and common methods such as readers and visualization functions. Each hazard class collects historical data or model simulations and transforms them, if necessary, in order to construct a coherent event database. Stochastic events can be generated taking into account the frequency and main intensity characteristics (such as local water depth for floods or gust speed for storms) of historical events, producing an ensemble of probabilistic events for each historical event. CLIMADA provides therefore an event-based probabilistic approach which does not depend on a hypothesis of a priori general probability distribution choices. Note that one can also reduce the probabilistic approach to a deterministic approach (e.g., story-line or forecasting) by defining the frequency to be 1. The source of the historical data (e.g. inventories or satellite images) or model simulations (e.g. synthetic tropical cyclone tracks) and the methodologies used to compute the hazard attributes and its stochastic events depend on each hazard type and are defined in its corresponding Hazard-derived class (e.g. `TropCylcone` for tropical cyclones, explained in the tutorial *TropCyclone*). This procedure provides a solid and homogeneous methodology to compute impacts worldwide. In the case where the risk analysis comprises a specific region where good quality data or models describing the hazard intensity and frequency are available, these can be directly ingested by the platform through the reader functions, skipping the hazard modelling part (in total or partially), and allowing us to easily and seamlessly combine CLIMADA with external sources. Hence the impact model can be used for a wide variety of applications, e.g. deterministically to assess the impact of a single (past or future) event or to quantify risk based on a (large) set of probabilistic events. Note that since the `Hazard` class is not an abstract class, any hazard that is not defined in CLIMADA can still be used by providing the `Hazard` attributes.

8.1.3 What do hazards look like in CLIMADA?

A `Hazard` contains events of some hazard type defined at `centroids`. There are certain variables in a `Hazard` instance that *are needed* to compute the impact, while others are *descriptive* and can therefore be set with default values. The full list of looks like this:

Mandatory variables	Data Type	Description
units	(str)	units of the intensity
centroids	`Centroids()`	centroids of the events
event_id	(np.array)	id (>0) of each event
frequency	(np.array)	frequency of each event in years
intensity	(sparse.csr_matrix)	intensity of the events at centroids
fraction	(sparse.csr_matrix)	fraction of affected exposures for each event at each centroid

Descriptive variables	Data Type	Description
date	(np.array	integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library)
orig	(np.array	flags indicating historical events (True) or probabilistic (False)
event_name	(list(str))	name of each event (default: event_id)

Note that `intensity` and `fraction` are scipy.sparse matrices of size num_events x num_centroids. The `fraction` attribute is optional. The `Centroids` class contains the geographical coordinates where the hazard is defined. A `Centroids` instance provides the coordinates either as points or raster data together with their Coordinate Reference System (CRS). The default CRS used in climada is the usual EPSG:4326. `Centroids` provides moreover methods to compute centroids areas, on land mask, country iso mask or distance to coast.

8.1.4 How is this tutorial structured?

Part 1: Read hazards from raster data

Part 2: Read hazards from other data

Part 3: Define hazards manually

Part 4: Analyse hazards

Part 5: Visualize hazards

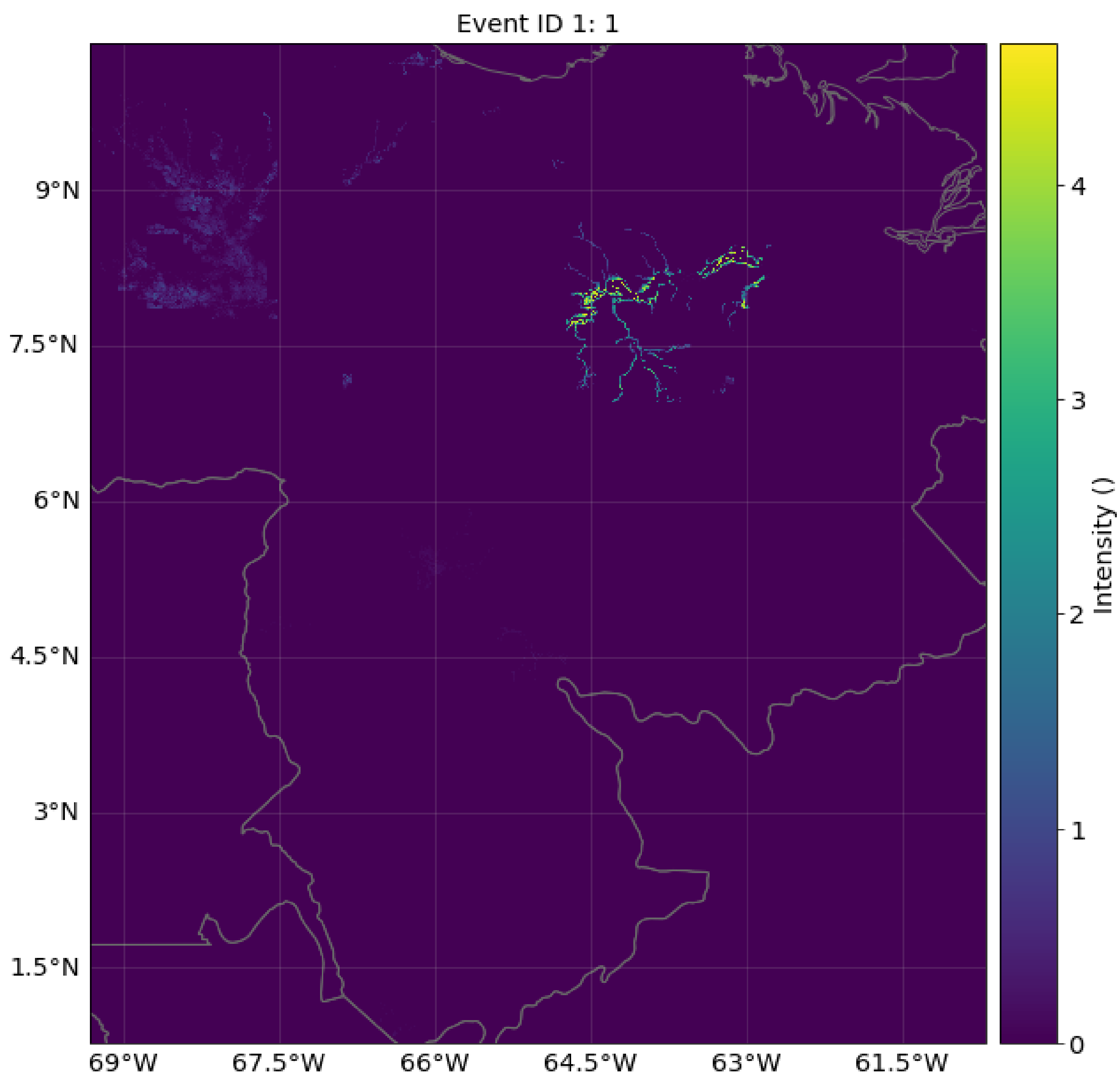
Part 6: Write (=save) hazards

Part 1: Read hazards from raster data

Raster data can be read in any format accepted by rasterio using `Hazard`'s `from_raster()` method. The raster information might refer to the `intensity` or `fraction`of the hazard. Different configuration options such as transforming the coordinates, changing the CRS and reading only a selected area or band are available through the `from_raster()` arguments as follows:

```
%matplotlib inline
import numpy as np
from climada.hazard import Hazard
from climada.util.constants import HAZ_DEMO_FL
# to hide the warnings
import warnings
warnings.filterwarnings('ignore')

# read intensity from raster file HAZ_DEMO_FL and set frequency for the contained event
haz_ven = Hazard.from_raster([HAZ_DEMO_FL], attrs={'frequency':np.ones(1)/2}, haz_
↪type='FL')
haz_ven.check()

# The masked values of the raster are set to 0
# Sometimes the raster file does not contain all the information, as in this case the␣
↪mask value -9999
# We mask it manuall and plot it using plot_intensity()
haz_ven.intensity[haz_ven.intensity==-9999] = 0
haz_ven.plot_intensity(1, smooth=False) # if smooth=True (default value) is used, the␣
↪computation time might increase

# per default the following attributes have been set
print('event_id: ', haz_ven.event_id)
print('event_name: ', haz_ven.event_name)
print('date: ', haz_ven.date)
print('frequency: ', haz_ven.frequency)
print('orig: ', haz_ven.orig)
print('min, max fraction: ', haz_ven.fraction.min(), haz_ven.fraction.max())
```

```
2022-03-16 22:09:56,965 - climada.util.coordinates - INFO - Reading C:\Users\yyljy\
↪climada\demo\data\SC22000_VE__M1.grd.gz
2022-03-16 22:10:01,099 - climada.util.coordinates - INFO - Reading C:\Users\yyljy\
↪climada\demo\data\SC22000_VE__M1.grd.gz
event_id:  [1]
event_name:  ['1']
date:  [1.]
frequency:  [0.5]
orig:  [ True]
min, max fraction:  0.0 1.0
```



8.1.5 EXERCISE:

1. Read raster data in EPSG 2201 Coordinate Reference System (CRS)
2. Read raster data in its given CRS and transform it to the affine transformation Affine(0.009000000000000341, 0.0, -69.33714959699981, 0.0, -0.009000000000000341, 10.42822096697894), height=500, width=501)
3. Read raster data in window Window(10, 10, 20, 30)

```
# Put your code here
```

```
# Solution:

# 1. The CRS can be reprojected using dst_crs option
haz = Hazard.from_raster([HAZ_DEMO_FL], dst_crs='epsg:2201', haz_type='FL')
```

(continues on next page)

(continued from previous page)

```
haz.check()
print('\n Solution 1:')
print('centroids CRS:', haz.centroids.crs)
print('raster info:', haz.centroids.meta)

# 2. Transformations of the coordinates can be set using the transform option and␣
↪Affine
from rasterio import Affine
haz = Hazard.from_raster([HAZ_DEMO_FL], haz_type='FL',
                         transform=Affine(0.009000000000000341, 0.0, -69.
↪33714959699981, \
                                          0.0, -0.009000000000000341, 10.
↪42822096697894),
                         height=500, width=501)
haz.check()
print('\n Solution 2:')
print('raster info:', haz.centroids.meta)
print('intensity size:', haz.intensity.shape)

# 3. A partial part of the raster can be loaded using the window or geometry
from rasterio.windows import Window
haz = Hazard.from_raster([HAZ_DEMO_FL], haz_type='FL', window=Window(10, 10, 20, 30))
haz.check()
print('\n Solution 3:')
print('raster info:', haz.centroids.meta)
print('intensity size:', haz.intensity.shape)
```

```
2022-03-16 22:11:00,548 - climada.util.coordinates - INFO - Reading C:\Users\yyljy\
↪climada\demo\data\SC22000_VE__M1.grd.gz
2022-03-16 22:11:04,941 - climada.util.coordinates - INFO - Reading C:\Users\yyljy\
↪climada\demo\data\SC22000_VE__M1.grd.gz

 Solution 1:
centroids CRS: epsg:2201
raster info: {'driver': 'GSBG', 'dtype': 'float32', 'nodata': 1.701410009187828e+38,
↪'width': 978, 'height': 1091, 'count': 1, 'crs': 'epsg:2201', 'transform':␣
↪Affine(1011.5372910988809, 0.0, 1120744.5486664253,
       0.0, -1011.5372910988809, 1189133.7652687666)}
2022-03-16 22:11:09,308 - climada.util.coordinates - INFO - Reading C:\Users\yyljy\
↪climada\demo\data\SC22000_VE__M1.grd.gz
2022-03-16 22:11:13,503 - climada.util.coordinates - INFO - Reading C:\Users\yyljy\
↪climada\demo\data\SC22000_VE__M1.grd.gz

 Solution 2:
raster info: {'driver': 'GSBG', 'dtype': 'float32', 'nodata': 1.701410009187828e+38,
↪'width': 501, 'height': 500, 'count': 1, 'crs': CRS.from_epsg(4326), 'transform':␣
↪Affine(0.009000000000000341, 0.0, -69.33714959699981,
       0.0, -0.009000000000000341, 10.42822096697894)}
intensity size: (1, 250500)
2022-03-16 22:11:17,739 - climada.util.coordinates - INFO - Reading C:\Users\yyljy\
↪climada\demo\data\SC22000_VE__M1.grd.gz
2022-03-16 22:11:17,780 - climada.util.coordinates - INFO - Reading C:\Users\yyljy\
↪climada\demo\data\SC22000_VE__M1.grd.gz

 Solution 3:
raster info: {'driver': 'GSBG', 'dtype': 'float32', 'nodata': 1.701410009187828e+38,
```

(continues on next page)

(continued from previous page)

```
↪'width': 20, 'height': 30, 'count': 1, 'crs': CRS.from_epsg(4326), 'transform':␣
↪Affine(0.009000000000000341, 0.0, -69.2471495969998,
       0.0, -0.009000000000000341, 10.338220966978936)}
intensity size: (1, 600)
```

Part 2: Read hazards from other data

- excel: Hazards can be read from Excel files following the template in `climada_python/climada/data/system/hazard_template.xlsx` using the `from_excel()` method.
- MATLAB: Hazards generated with CLIMADA's MATLAB version (.mat format) can be read using `from_mat()`.
- vector data: Use `Hazard`'s `from_vector`-constructor to read shape data (all formats supported by fiona).
- hdf5: Hazards generated with the CLIMADA in Python (.h5 format) can be read using `from_hdf5()`.

```
from climada.hazard import Hazard, Centroids
from climada.util import HAZ_DEMO_H5 # CLIMADA's Python file
# Hazard needs to know the acronym of the hazard type to be constructed!!! Use 'NA'␣
↪if not known.
haz_tc_fl = Hazard.from_hdf5(HAZ_DEMO_H5)  # Historic tropical cyclones in Florida␣
↪from 1990 to 2004
haz_tc_fl.check() # Use always the check() method to see if the hazard has been␣
↪loaded correctly
```

```
2022-03-16 22:11:26,805 - climada.hazard.base - INFO - Reading C:\Users\yyljy\climada\
↪demo\data\tc_fl_1990_2004.h5
```

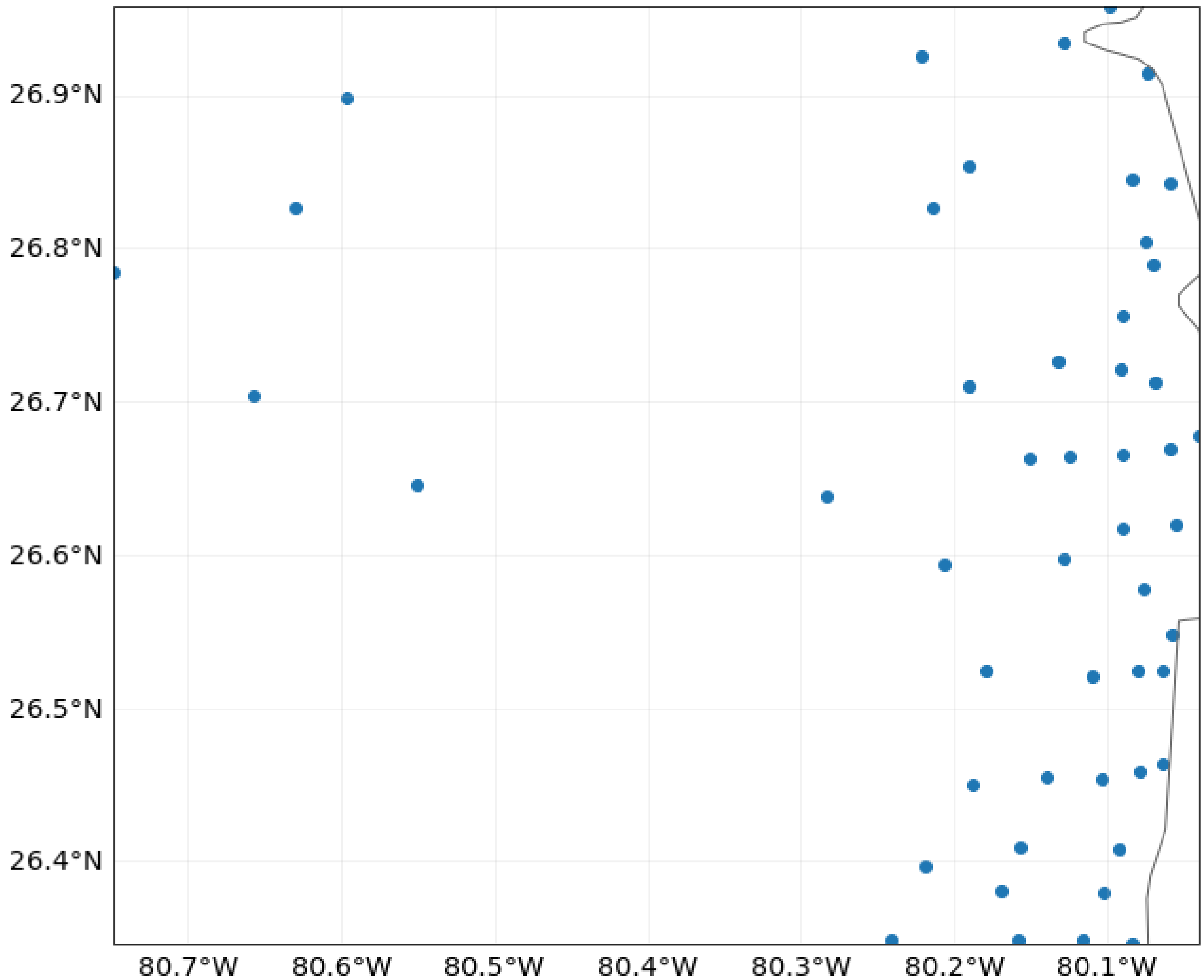
Part 3: Define hazards manually A `Hazard` can be defined by filling its values one by one, as follows:

```
# setting points
import numpy as np
from scipy import sparse

lat = np.array([26.933899, 26.957203, 26.783846, 26.645524, 26.897796, 26.925359, \
       26.914768, 26.853491, 26.845099, 26.82651 , 26.842772, 26.825905, \
       26.80465 , 26.788649, 26.704277, 26.71005 , 26.755412, 26.678449, \
       26.725649, 26.720599, 26.71255 , 26.6649  , 26.664699, 26.663149, \
       26.66875 , 26.638517, 26.59309 , 26.617449, 26.620079, 26.596795, \
       26.577049, 26.524585, 26.524158, 26.523737, 26.520284, 26.547349, \
       26.463399, 26.45905 , 26.45558 , 26.453699, 26.449999, 26.397299, \
       26.4084  , 26.40875 , 26.379113, 26.3809  , 26.349068, 26.346349, \
       26.348015, 26.347957])

lon = np.array([-80.128799, -80.098284, -80.748947, -80.550704, -80.596929, \
       -80.220966, -80.07466 , -80.190281, -80.083904, -80.213493, \
       -80.0591  , -80.630096, -80.075301, -80.069885, -80.656841, \
       -80.190085, -80.08955 , -80.041179, -80.1324  , -80.091746, \
       -80.068579, -80.090698, -80.1254  , -80.151401, -80.058749, \
       -80.283371, -80.206901, -80.090649, -80.055001, -80.128711, \
       -80.076435, -80.080105, -80.06398 , -80.178973, -80.110519, \
       -80.057701, -80.064251, -80.07875 , -80.139247, -80.104316, \
       -80.188545, -80.21902 , -80.092391, -80.1575  , -80.102028, \
```

(continues on next page)

(continued from previous page)

```
       -80.16885 , -80.116401, -80.08385 , -80.241305, -80.158855])

n_cen = lon.size # number of centroids
n_ev = 10 # number of events

intensity = sparse.csr_matrix(np.random.random((n_ev, n_cen)))
fraction = intensity.copy()
fraction.data.fill(1)

haz = Hazard(haz_type='TC',
             intensity=intensity,
             fraction=fraction,
             centroids=Centroids.from_lat_lon(lat, lon),  # default crs used
             units='m',
             event_id=np.arange(n_ev, dtype=int),
             event_name=['ev_12', 'ev_21', 'Maria', 'ev_35',
                         'Irma', 'ev_16', 'ev_15', 'Edgar', 'ev_1', 'ev_9'],
             date=np.array([721166, 734447, 734447, 734447, 721167,
                            721166, 721167, 721200, 721166, 721166]),
             orig=np.zeros(n_ev, bool),
             frequency=np.ones(n_ev)/n_ev,)

haz.check()
haz.centroids.plot();
```



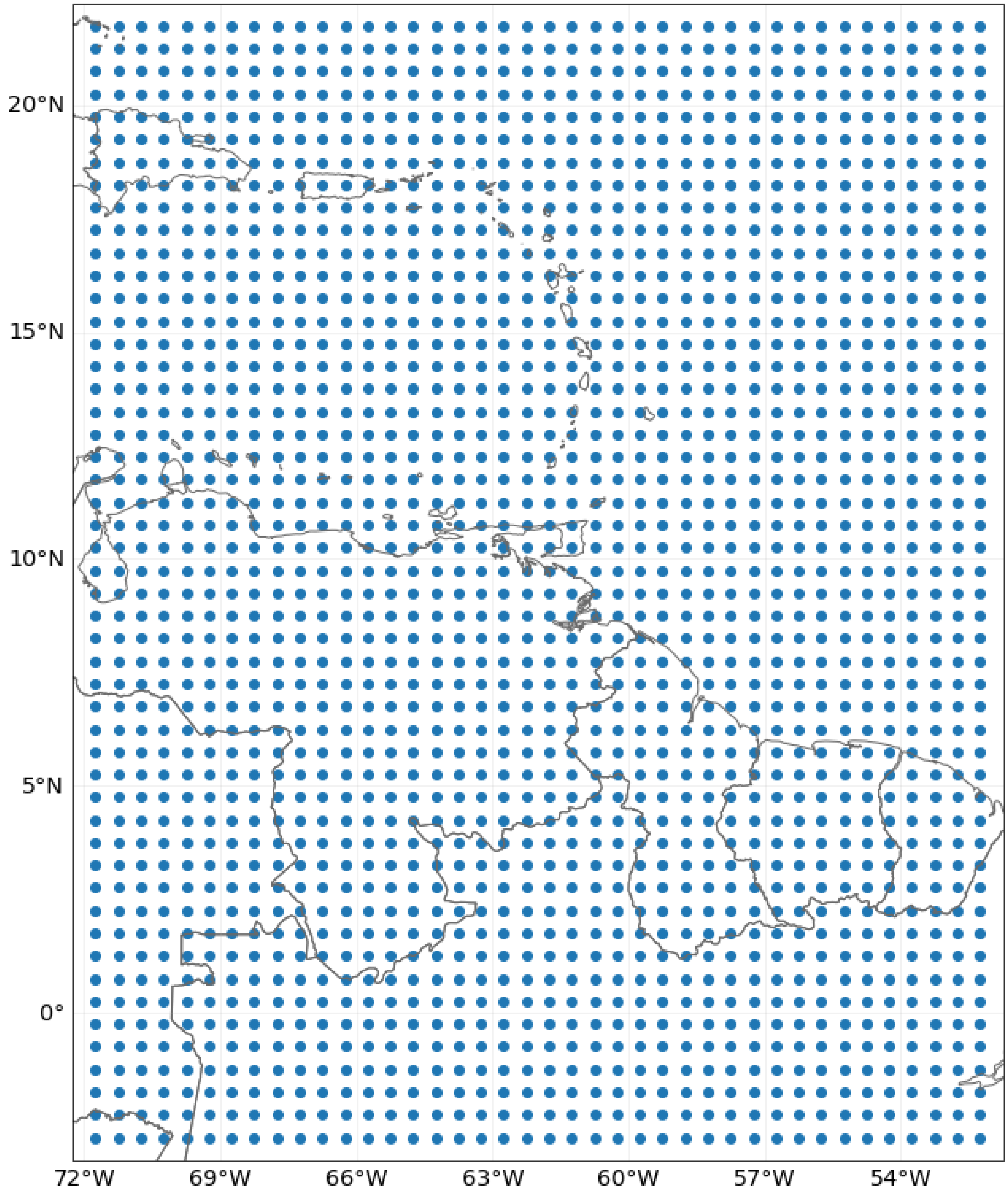

```
# setting raster
import numpy as np
from scipy import sparse

# raster info:
# border upper left corner (of the pixel, not of the center of the pixel)
xf_lat = 22
xo_lon = -72
# resolution in lat and lon
d_lat = -0.5 # negative because starting in upper corner
d_lon = 0.5 # same step as d_lat
# number of points
n_lat = 50
n_lon = 40

n_ev = 10 # number of events
centroids = Centroids.from_pix_bounds(xf_lat, xo_lon, d_lat, d_lon, n_lat, n_lon) #␣
↪default crs used
intensity = sparse.csr_matrix(np.random.random((n_ev, centroids.size)))
fraction = intensity.copy()
fraction.data.fill(1)
```

(continues on next page)

(continued from previous page)

```
haz = Hazard('TC',
             centroids=centroids,
             intensity=intensity,
             fraction=fraction,
             units='m',
             event_id=np.arange(n_ev, dtype=int),
             event_name=['ev_12', 'ev_21', 'Maria', 'ev_35',
                         'Irma', 'ev_16', 'ev_15', 'Edgar', 'ev_1', 'ev_9'],
             date=np.array([721166, 734447, 734447, 734447, 721167,
                            721166, 721167, 721200, 721166, 721166]),
             orig=np.zeros(n_ev, bool),
             frequency=np.ones(n_ev)/n_ev,)

haz.check()
print('Check centroids borders:', haz.centroids.total_bounds)
haz.centroids.plot()

# using from_pnt_bounds, the bounds refer to the bounds of the center of the pixel
left, bottom, right, top = xo_lon, -3.0, -52.0, xf_lat
haz.centroids = Centroids.from_pnt_bounds((left, bottom, right, top), 0.5) # default␣
↪crs used
print('Check centroids borders:', haz.centroids.total_bounds)
```

```
Check centroids borders: (-72.0, -3.0, -52.0, 22.0)
Check centroids borders: (-72.25, -3.25, -51.75, 22.25)
```


20°N
15°N
10°N
5°N
0°
72°W
69°W
66°W
63°W
60°W
57°W
54°W

8.1.6 Part 4: Analyse Hazards

The following methods can be used to analyse the data in `Hazard`:

- `calc_year_set()` method returns a dictionary with all the historical (not synthetic) event ids that happened at each year.
- `get_event_date()` returns strings of dates in ISO format.
- To obtain the relation between event ids and event names, two methods can be used `get_event_name()` and `get_event_id()`.

Other methods to handle one or several Hazards are:

- the property `size` returns the number of events contained.
- `append()` is used to expand events with data from another `Hazard` (and same centroids).
- `select()` returns a new hazard with the selected region, date and/or synthetic or historical filter.
- `remove_duplicates()` removes events with same name and date.
- `local_exceedance_inten()` returns a matrix with the exceedence frequency at every frequency and provided return periods. This is the one used in `plot_rp_intensity()`.
- `reproject_raster()`, `reproject_vector()`, `raster_to_vector()`, `vector_to_raster()` are methods to change centroids' CRS and between raster and vector data.

`Centroids` methods:

- centroids properties such as area per pixel, distance to coast, country ISO code, on land mask or elevation are available through different `set_XX()` methods.
- `set_lat_lon_to_meta()` computes the raster `meta` dictionary from present lat and lon. `set_meta_to_lat_lon()` computes lat and lon of the center of the pixels described in attribute `meta`. The raster `meta` information contains at least: `width`, `height`, `crs` and `transform` data (use help(Centroids) for more info). Using raster centroids can increase computing performance for several computations.
- when using lats and lons (vector data) the `geopandas.GeoSeries geometry` attribute contains the CRS information and can be filled with point shapes to perform different computation. The geometry points can be then released using `empty_geometry_points()`.

EXERCISE:

Using the previous hazard `haz_tc_fl` answer these questions:

1. How many synthetic events are contained?
2. Generate a hazard with historical hurricanes ocurring between 1995 and 2001.
3. How many historical hurricanes occured in 1999? Which was the year with most hurricanes between 1995 and 2001?
4. What is the number of centroids with distance to coast smaller than 1km?

```
# Put your code here:
```

```
#help(hist_tc.centroids) # If you want to run it, do it after you execute the next␣
↪block
```

```
# SOLUTION:

# 1.How many synthetic events are contained?
print('Number of total events:', haz_tc_fl.size)
print('Number of synthetic events:', np.logical_not(haz_tc_fl.orig).astype(int).sum())

# 2. Generate a hazard with historical hurricanes ocurring between 1995 and 2001.
hist_tc = haz_tc_fl.select(date=('1995-01-01', '2001-12-31'), orig=True)
print('Number of historical events between 1995 and 2001:', hist_tc.size)

# 3. How many historical hurricanes occured in 1999? Which was the year with most␣
↪hurricanes between 1995 and 2001?
ev_per_year = hist_tc.calc_year_set() # events ids per year
print('Number of events in 1999:', ev_per_year[1999].size)
max_year = 1995
max_ev = ev_per_year[1995].size
for year, ev in ev_per_year.items():
    if ev.size > max_ev:
        max_year = year
print('Year with most hurricanes between 1995 and 2001:', max_year)

# 4. What is the number of centroids with distance to coast smaller than 1km?
hist_tc.centroids.set_dist_coast()
num_cen_coast = np.argwhere(hist_tc.centroids.dist_coast < 1000).size
print('Number of centroids close to coast: ', num_cen_coast)
```

```
Number of total events: 216
Number of synthetic events: 0
Number of historical events between 1995 and 2001: 109
Number of events in 1999: 16
Year with most hurricanes between 1995 and 2001: 1995
2022-03-16 22:12:55,499 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2022-03-16 22:12:57,355 - climada.util.coordinates - INFO - dist_to_coast: UTM 32617␣
↪(1/2)
2022-03-16 22:12:59,926 - climada.util.coordinates - INFO - dist_to_coast: UTM 32618␣
↪(2/2)
Number of centroids close to coast:  41
```

8.1.7 Part 5: Visualize Hazards

There are three different plot functions: `plot_intensity()`, `plot_fraction()` and `plot_rp_intensity()`. Depending on the inputs, different properties can be visualized. Check the documentation of the functions:

```
help(haz_tc_fl.plot_intensity)
help(haz_tc_fl.plot_rp_intensity)
```

```
Help on method plot_intensity in module climada.hazard.base:

plot_intensity(event=None, centr=None, smooth=True, axis=None, adapt_fontsize=True,␣
↪**kwargs) method of climada.hazard.base.Hazard instance
    Plot intensity values for a selected event or centroid.
```

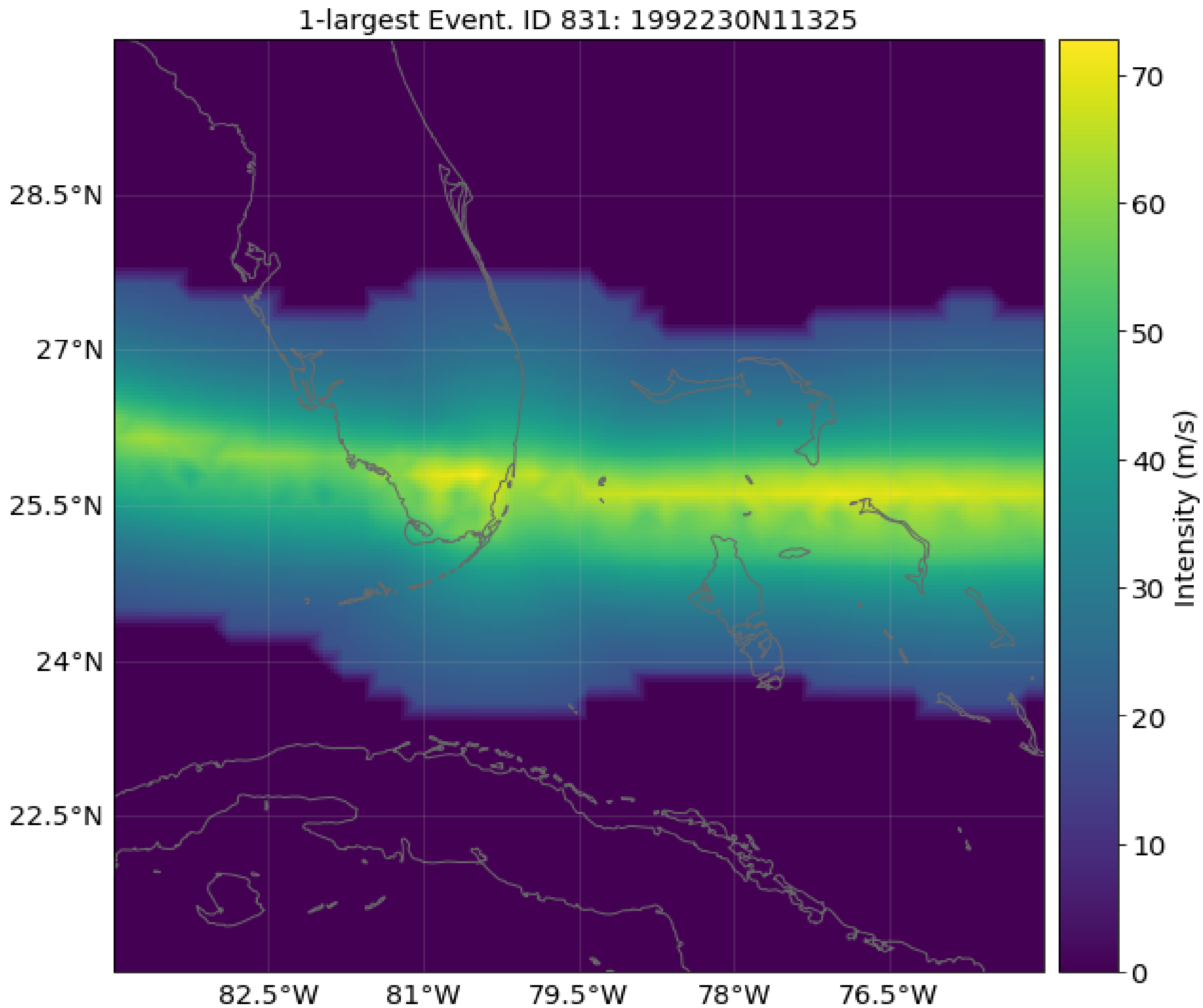
(continues on next page)

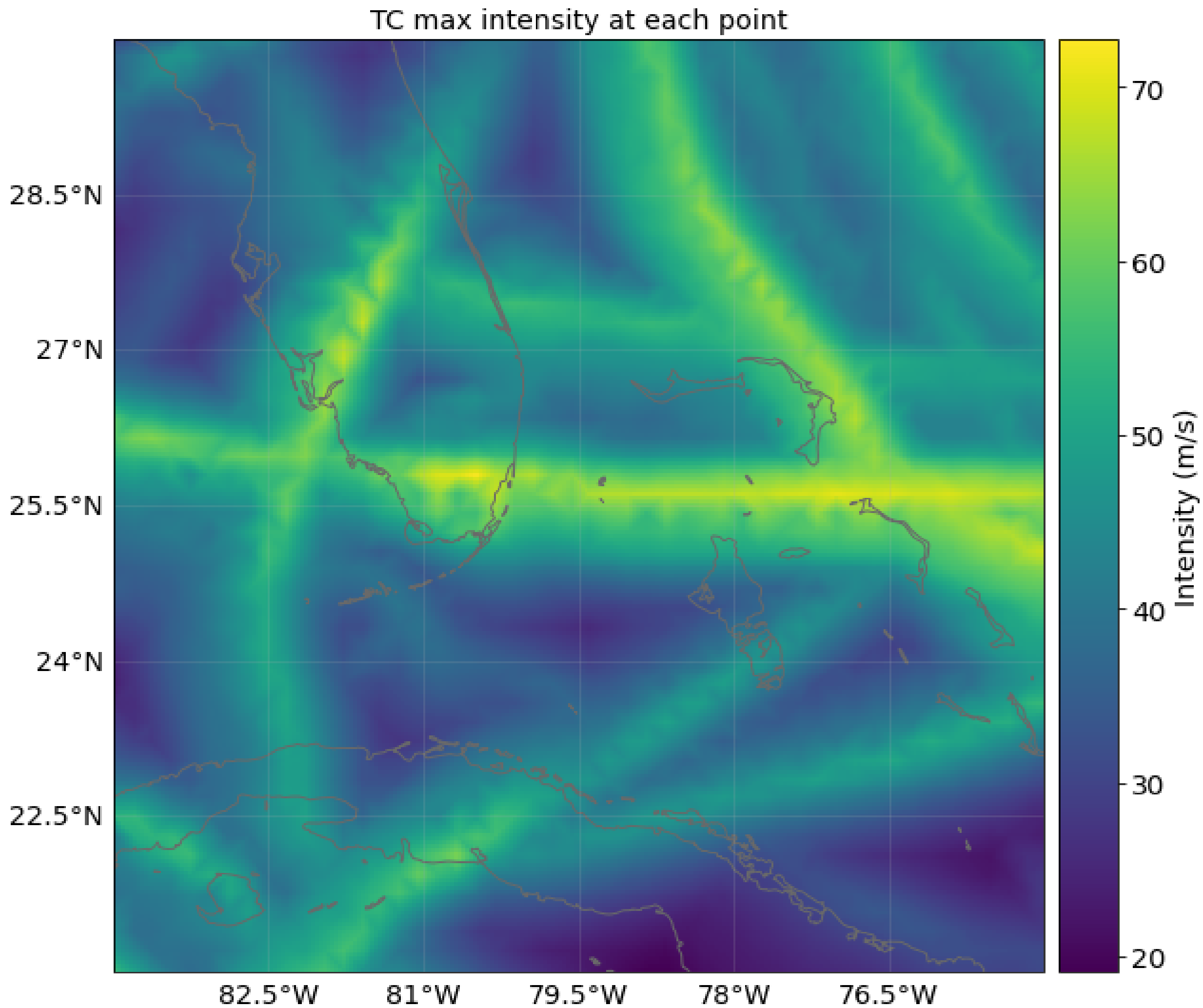
(continued from previous page)

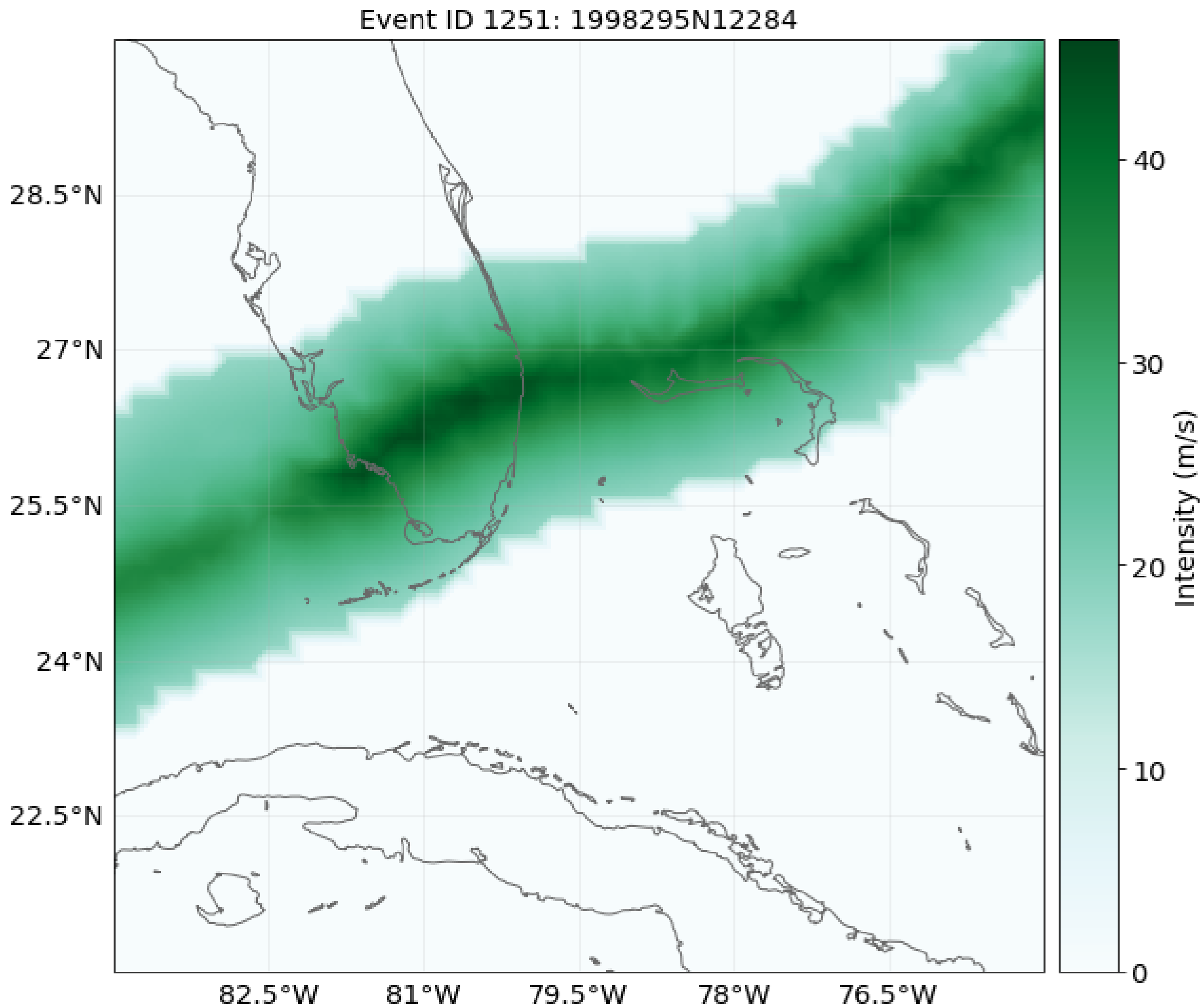
```
    Parameters
    ----------
    event: int or str, optional
        If event > 0, plot intensities of
        event with id = event. If event = 0, plot maximum intensity in
        each centroid. If event < 0, plot abs(event)-largest event. If
        event is string, plot events with that name.
    centr: int or tuple, optional
        If centr > 0, plot intensity
        of all events at centroid with id = centr. If centr = 0,
        plot maximum intensity of each event. If centr < 0,
        plot abs(centr)-largest centroid where higher intensities
        are reached. If tuple with (lat, lon) plot intensity of nearest
        centroid.
    smooth: bool, optional
        Rescale data to RESOLUTIONxRESOLUTION pixels (see constant
        in module `climada.util.plot`)
    axis: matplotlib.axes._subplots.AxesSubplot, optional
        axis to use
    kwargs: optional
        arguments for pcolormesh matplotlib function
        used in event plots or for plot function used in centroids plots

    Returns
    -------
        matplotlib.axes._subplots.AxesSubplot

    Raises
    ------
        ValueError

Help on method plot_rp_intensity in module climada.hazard.base:

plot_rp_intensity(return_periods=(25, 50, 100, 250), smooth=True, axis=None,␣
↪figsize=(9, 13), adapt_fontsize=True, **kwargs) method of climada.hazard.base.
↪Hazard instance
    Compute and plot hazard exceedance intensity maps for different
    return periods. Calls local_exceedance_inten.

    Parameters
    ----------
    return_periods: tuple(int), optional
        return periods to consider
    smooth: bool, optional
        smooth plot to plot.RESOLUTIONxplot.RESOLUTION
    axis: matplotlib.axes._subplots.AxesSubplot, optional
        axis to use
    figsize: tuple, optional
        figure size for plt.subplots
    kwargs: optional
        arguments for pcolormesh matplotlib function used in event plots

    Returns
    -------
    axis, inten_stats:  matplotlib.axes._subplots.AxesSubplot, np.ndarray
        intenstats is return_periods.size x num_centroids
```

```
# 1. intensities of the largest event (defined as greater sum of intensities):
# all events:
haz_tc_fl.plot_intensity(event=-1) # largest historical event: 1992230N11325␣
↪hurricane ANDREW

# 2. maximum intensities at each centroid:
haz_tc_fl.plot_intensity(event=0)

# 3. intensities of hurricane 1998295N12284:
haz_tc_fl.plot_intensity(event='1998295N12284', cmap='BuGn') # setting color map

# 4. tropical cyclone intensities maps for the return periods [10, 50, 75, 100]
_, res = haz_tc_fl.plot_rp_intensity([10, 50, 75, 100])

# 5. intensities of all the events in centroid with id 50
haz_tc_fl.plot_intensity(centr=50)

# 6. intensities of all the events in centroid closest to lat, lon = (26.5, -81)
haz_tc_fl.plot_intensity(centr=(26.5, -81));
```

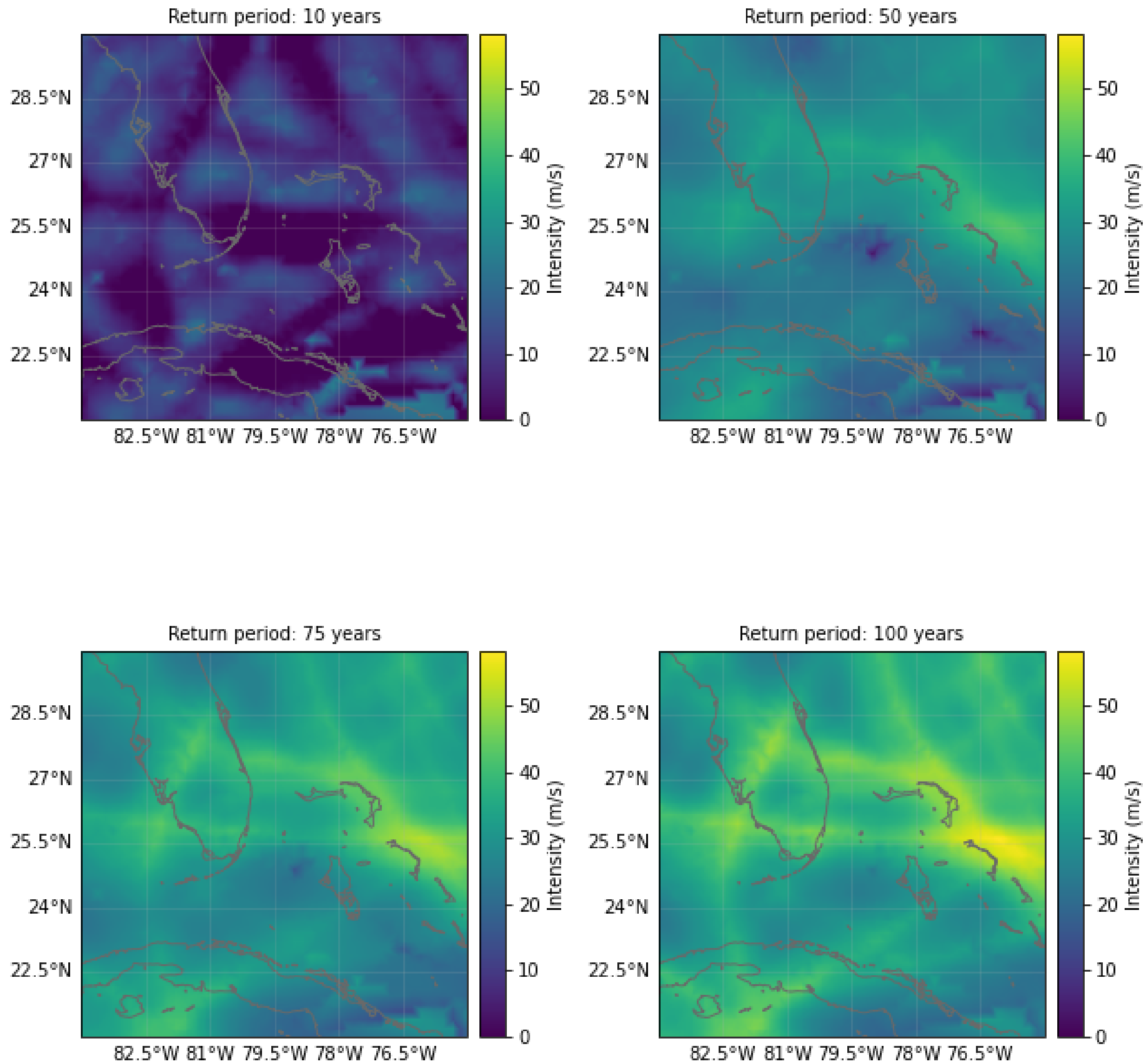
```
2022-03-16 22:13:50,822 - climada.hazard.base - INFO - Computing exceedance intenstiy␣
↪map for return periods: [ 10  50  75 100]
2022-03-16 22:13:51,449 - climada.hazard.base - WARNING - Exceedance intenstiy values␣
↪below 0 are set to 0.                    Reason: no negative intensity values were␣
↪found in hazard.
```


1-largest Event. ID 831: 1992230N11325
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
Intensity (m/s)
0
10
20
30
40
50
60
70

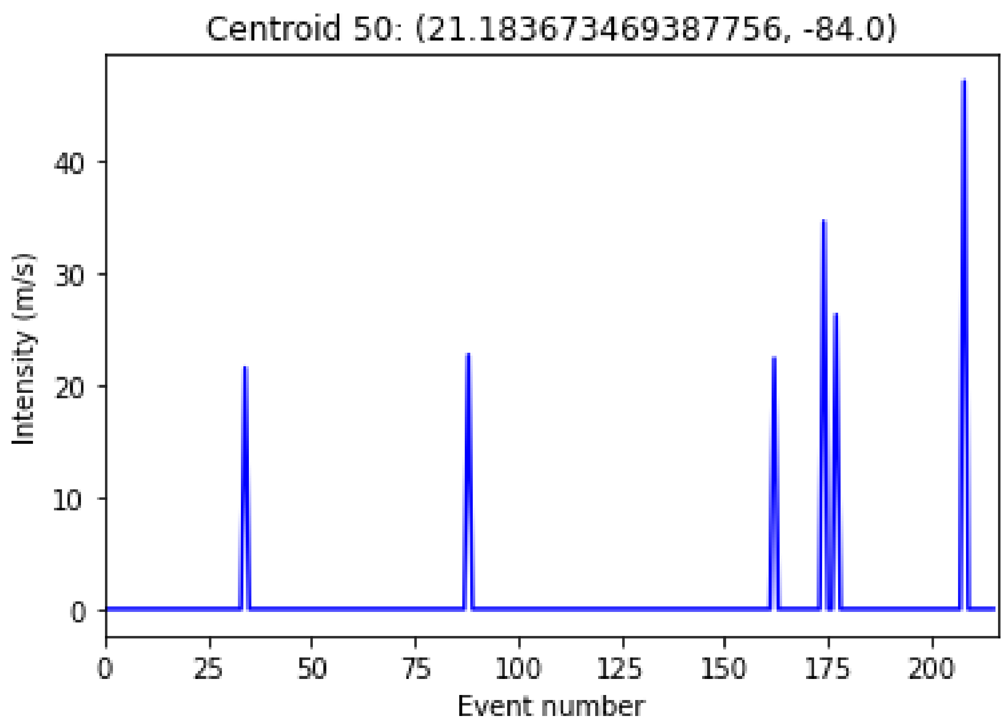

TC max intensity at each point
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
70
60
50
40
30
20
Intensity (m/s)

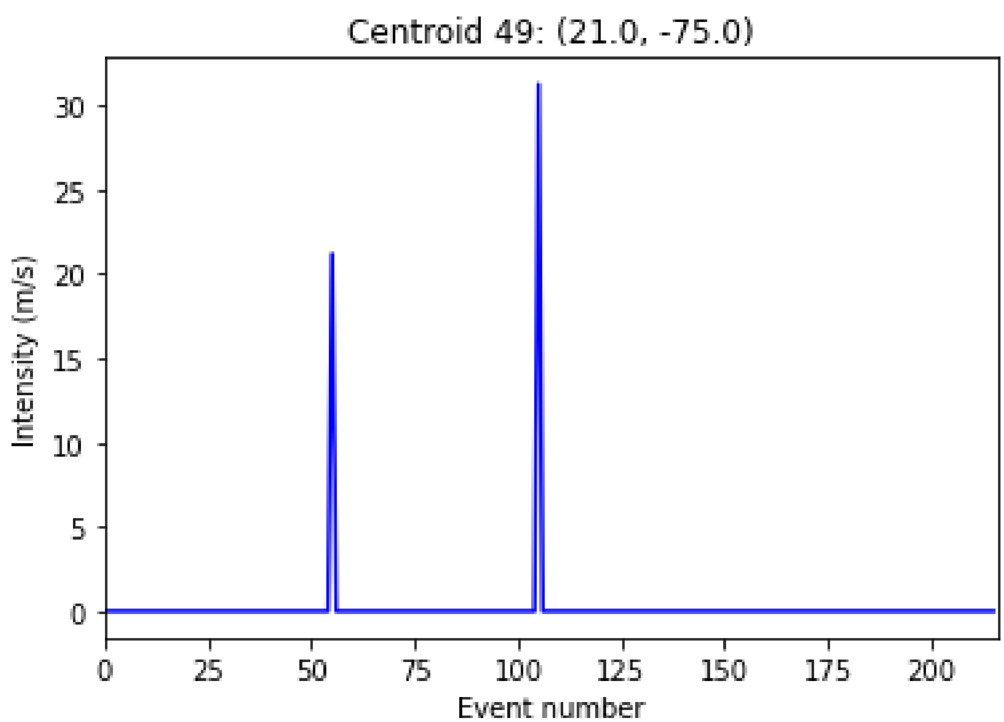


Event ID 1251: 1998295N12284
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
Intensity (m/s)
40
30
20
10
0



Return period: 10 years
Return period: 50 years
Return period: 75 years
Return period: 100 years
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W 81°W 79.5°W 78°W 76.5°W
Intensity (m/s)
0
10
20
30
40
50



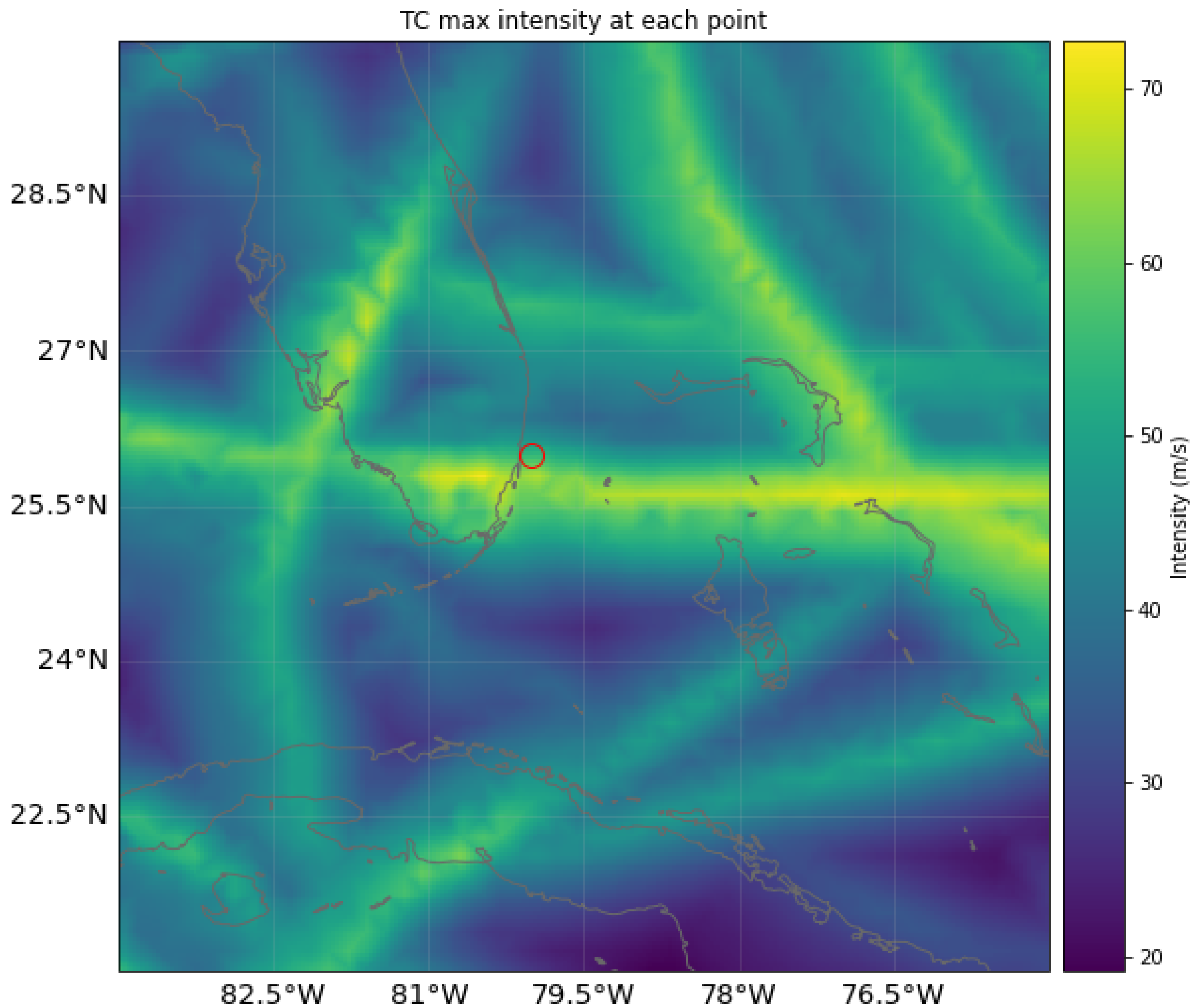



```
# 7. one figure with two plots: maximum intensities and selected centroid with all␣
↪intensities:
from climada.util.plot import make_map
import matplotlib.pyplot as plt
plt.ioff()
fig, ax1, fontsize = make_map(1)  # map
```

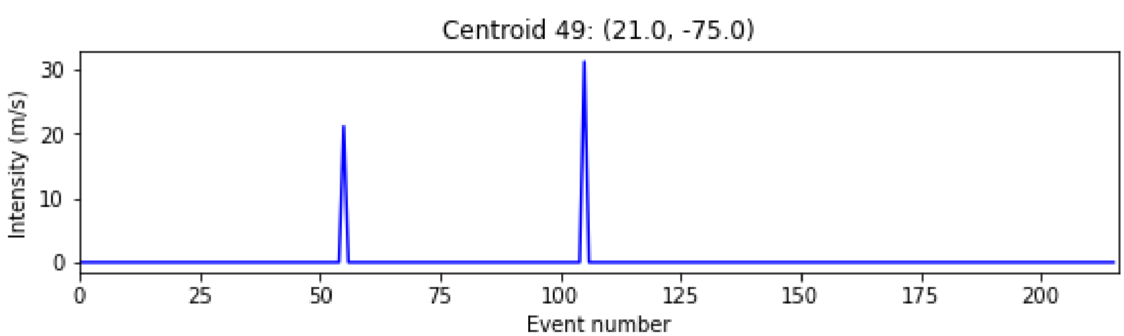
(continues on next page)

(continued from previous page)

```
ax2 = fig.add_subplot(2, 1, 2) # add regular axes
haz_tc_fl.plot_intensity(axis=ax1, event=0) # plot original resolution
ax1.plot(-80, 26, 'or', mfc='none', markersize=12)
haz_tc_fl.plot_intensity(axis=ax2, centr=(26, -80))
fig.subplots_adjust(hspace=6.5)
```



TC max intensity at each point
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
70
60
50
40
30
20
Intensity (m/s)



Centroid 49: (21.0, -75.0)
Intensity (m/s)
30
20
10
0
0
25
50
75
100
125
150
175
200
Event number

8.1.8 Part 6: Write (=save) hazards

Hazards can be written and read in hdf5 format as follows:

```
haz_tc_fl.write_hdf5('results/haz_tc_fl.h5')

haz = Hazard.from_hdf5('results/haz_tc_fl.h5')
haz.check()
```

```
2022-03-16 22:17:30,344 - climada.hazard.base - INFO - Writing results/haz_tc_fl.h5
2022-03-16 22:17:30,572 - climada.hazard.base - INFO - Reading results/haz_tc_fl.h5
```

GeoTiff data is generated using `write_raster()`:

```
haz_ven.write_raster('results/haz_ven.tif') # each event is a band of the tif file
```

```
2022-03-16 22:17:41,283 - climada.util.coordinates - INFO - Writting results/haz_ven.
↪tif
```

Pickle will work as well:

```
from climada.util.save import save
# this generates a results folder in the current path and stores the output there
save('tutorial_haz_tc_fl.p', haz_tc_fl)
```

```
2022-03-16 22:17:55,836 - climada.util.save - INFO - Written file C:\Users\yyljy\
↪Documents\climada_main\doc\tutorial\results\tutorial_haz_tc_fl.p
```

8.2 Hazard: Tropical cyclones

Tropical cyclones tracks are gathered in the class `TCTracks` and then provided to the hazard `TropCyclone` which computes the wind gusts at each centroid. `TropCyclone` inherits from `Hazard` and has an associated hazard type `TC`.

8.2.1 What do tropical cyclones look like in CLIMADA?

`TCTracks` reads and handles historical tropical cyclone tracks of the IBTrACS repository or synthetic tropical cyclone tracks simulated using fully statistical or coupled statistical-dynamical modeling approaches. It also generates synthetic tracks from the historical ones using Wiener processes.

The tracks are stored in the attribute `data`, which is a list of `xarray`'s `Dataset` (see xarray.Dataset). Each `Dataset` contains the following variables:

Coordinates
time
latitude
longitude

Descriptive variables
time_step
radius_max_wind
max_sustained_wind
central_pressure
environmental_pressure

Attributes
max_sustained_wind_unit
central_pressure_unit
sid
name
orig_event_flag
data_provider
basin
id_no
category

8.2.2 How is this tutorial structured?

Part 1: Load TC tracks

a) Load TC tracks from historical records

b) Generate probabilistic events

c) ECMWF Forecast Tracks

d) Load TC tracks from other sources

Part 2: `TropCyclone()` class

a) Default hazard generation for tropical cyclones

b) Implementing climate change

c) Multiprocessing - improving performance for big computations

d) Making videos

8.2.3 Part 1: Load TC tracks

Records of historical TCs are very limited and therefore the database to study this natural hazard remains sparse. Only a small fraction of the TCs make landfall every year and reliable documentation of past TC landfalling events has just started in the 1950s (1980s - satellite era). The generation of synthetic storm tracks is an important tool to overcome this spatial and temporal limitation. Synthetic dataset are much larger and thus allow to estimate the risk of much rarer events. Here we show the most prominent tools in CLIMADA to load TC tracks from historical records *a)*, generate a probabilistic dataset thereof *b)*, and work with model simulations *c)*.

8.2.4 a) Load TC tracks from historical records

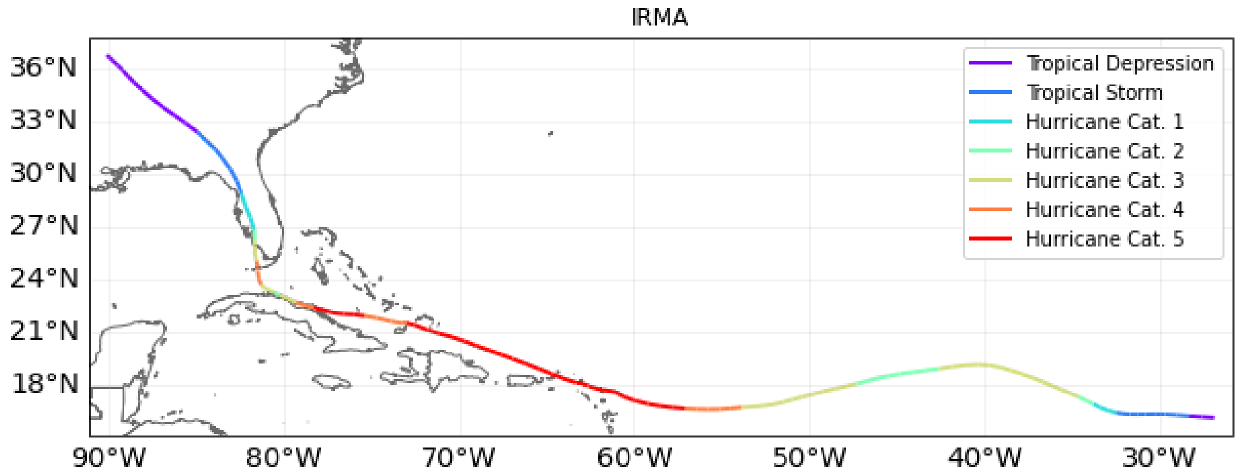
The best-track historical data from the International Best Track Archive for Climate Stewardship (IBTrACS) can easily be loaded into CLIMADA to study the historical records of TC events. The constructor `from_ibtracs_netcdf()` generates the `Datasets` for tracks selected by IBTrACS id, or by basin and year range. To achieve this, it downloads the first time the IBTrACS data v4 in netcdf format and stores it in `~/climada/data/`. The tracks can be accessed later either using the attribute `data` or using `get_track()`, which allows to select tracks by its name or id. Use the method `append()` to extend the `data` list.

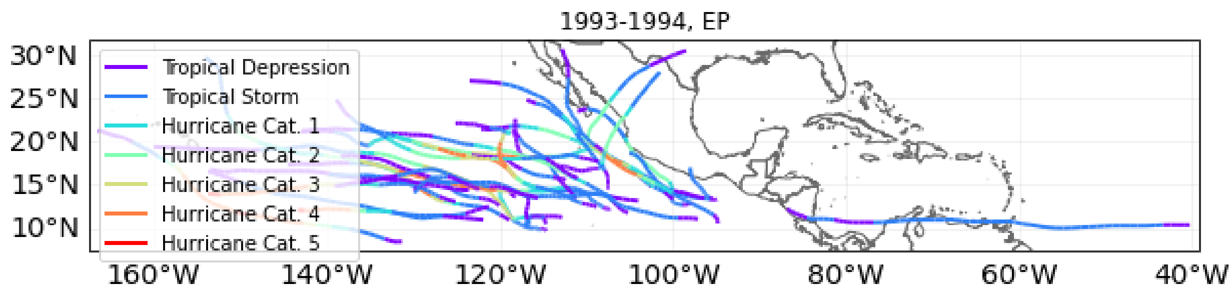
If you get an error downloading the IBTrACS data, try to manually access https://www.ncei.noaa.gov/data/international-best-track-archive-for-climate-stewardship-ibtracs/v04r00/access/netcdf/, click on the file `IBTrACS.ALL.v04r00.nc` and copy it to `~/climada/data/`.

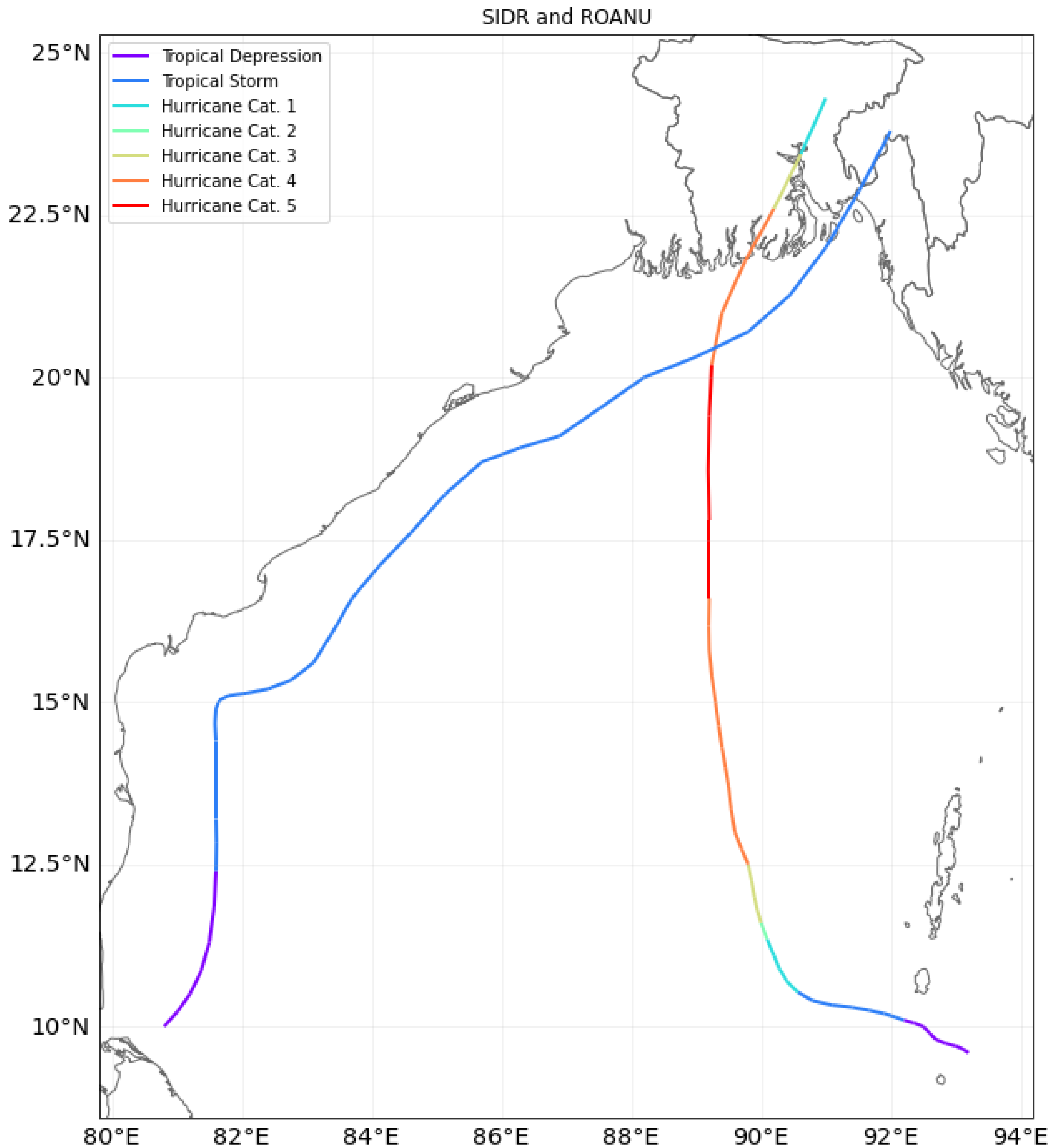
To visualize the tracks use `plot()`.

```
%matplotlib inline
from climada.hazard import TCTracks

tr_irma = TCTracks.from_ibtracs_netcdf(provider='usa', storm_id='2017242N16333') #␣
↪IRMA 2017
ax = tr_irma.plot();
ax.set_title('IRMA') # set title

# other ibtracs selection options
from climada.hazard import TCTracks
# years 1993 and 1994 in basin EP.
# correct_pres ignores tracks with not enough data. For statistics (frequency of␣
↪events), these should be considered as well
sel_ibtracs = TCTracks.from_ibtracs_netcdf(provider='usa', year_range=(1993, 1994),␣
↪basin='EP', correct_pres=False)
print('Number of tracks:', sel_ibtracs.size)
ax = sel_ibtracs.plot();
ax.get_legend()._loc = 2 # correct legend location
ax.set_title('1993-1994, EP') # set title

track1 = TCTracks.from_ibtracs_netcdf(provider='usa', storm_id='2007314N10093') #␣
↪SIDR 2007
track2 = TCTracks.from_ibtracs_netcdf(provider='usa', storm_id='2016138N10081') #␣
↪ROANU 2016
track1.append(track2.data) # put both tracks together
ax = track1.plot();
ax.get_legend()._loc = 2 # correct legend location
ax.set_title('SIDR and ROANU'); # set title
```

```
2021-06-04 17:07:33,515 - climada.hazard.tc_tracks - INFO - Progress: 100%
2021-06-04 17:07:35,833 - climada.hazard.tc_tracks - WARNING - 19 storm events are␣
↪discarded because no valid wind/pressure values have been found: 1993178N14265,␣
↪1993221N12216, 1993223N07185, 1993246N16129, 1993263N11168, ...
2021-06-04 17:07:35,940 - climada.hazard.tc_tracks - INFO - Progress: 11%
2021-06-04 17:07:36,028 - climada.hazard.tc_tracks - INFO - Progress: 23%
2021-06-04 17:07:36,119 - climada.hazard.tc_tracks - INFO - Progress: 35%
2021-06-04 17:07:36,218 - climada.hazard.tc_tracks - INFO - Progress: 47%
2021-06-04 17:07:36,312 - climada.hazard.tc_tracks - INFO - Progress: 58%
2021-06-04 17:07:36,399 - climada.hazard.tc_tracks - INFO - Progress: 70%
2021-06-04 17:07:36,493 - climada.hazard.tc_tracks - INFO - Progress: 82%
2021-06-04 17:07:36,585 - climada.hazard.tc_tracks - INFO - Progress: 94%
2021-06-04 17:07:36,612 - climada.hazard.tc_tracks - INFO - Progress: 100%
Number of tracks: 33
2021-06-04 17:07:38,825 - climada.hazard.tc_tracks - INFO - Progress: 100%
2021-06-04 17:07:39,974 - climada.hazard.tc_tracks - INFO - Progress: 100%
```








```
tr_irma.get_track('2017242N16333')
```

```
<xarray.Dataset>
Dimensions:                    (time: 123)
Coordinates:
  * time                       (time) datetime64[ns] 2017-08-30 ... 2017-09-13T1...
    lat                        (time) float32 16.1 16.15 16.2 ... 36.2 36.5 36.8
    lon                        (time) float32 -26.9 -27.59 -28.3 ... -89.79 -90.1
Data variables:
    time_step                  (time) float64 3.0 3.0 3.0 3.0 ... 3.0 3.0 3.0 3.0
```

(continues on next page)

(continued from previous page)

```
    radius_max_wind          (time) float32 60.0 60.0 60.0 ... 60.0 60.0 60.0
    radius_oci               (time) float32 180.0 180.0 180.0 ... 350.0 350.0
    max_sustained_wind       (time) float32 30.0 32.0 35.0 ... 15.0 15.0 15.0
    central_pressure         (time) float32 1.008e+03 1.007e+03 ... 1.005e+03
    environmental_pressure   (time) float64 1.012e+03 1.012e+03 ... 1.008e+03
    basin                    (time) <U2 'NA' 'NA' 'NA' 'NA' ... 'NA' 'NA' 'NA'
Attributes:
    max_sustained_wind_unit:  kn
    central_pressure_unit:    mb
    name:                     IRMA
    sid:                      2017242N16333
    orig_event_flag:          True
    data_provider:            ibtracs_usa
    id_no:                    2017242016333.0
    category:                 5
```

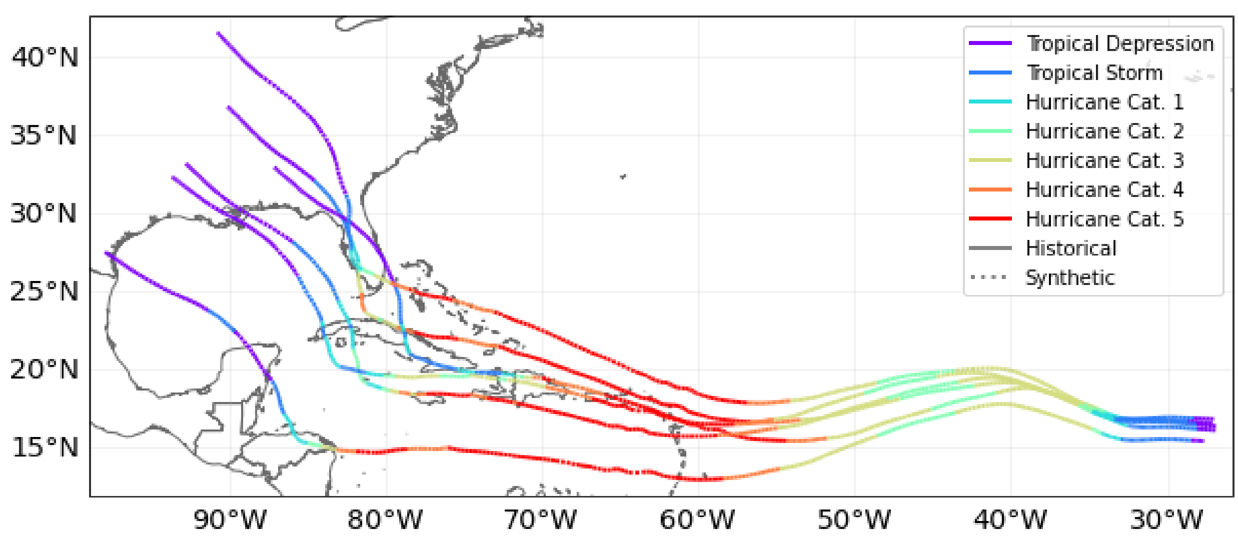
8.2.5 b) Generate probabilistic events

Once tracks are present in `TCTracks`, one can generate synthetic tracks for each present track based on directed random walk. Note that the tracks should be interpolated to use the same timestep **before** generation of probabilistic events.

`calc_perturbed_trajectories()` generates an ensemble of "nb_synth_tracks" numbers of synthetic tracks is computed for every track. The methodology perturbs the tracks locations, and if decay is `True` it additionally includes decay of wind speed and central pressure drop after landfall. No other track parameter is perturbed.

```
# here we use tr_irma retrieved from IBTrACS with the function above
# select number of synthetic tracks (nb_synth_tracks) to generate per present tracks.
tr_irma.equal_timestep()
tr_irma.calc_perturbed_trajectories(nb_synth_tracks=5)
tr_irma.plot();
# see more configutration options (e.g. amplitude of max random starting point shift␣
↪in decimal degree; max_shift_ini)
```

```
<GeoAxesSubplot:>
```



```
tr_irma.data[-1] # last synthetic track. notice the value of orig_event_flag and name
```

```
<xarray.Dataset>
Dimensions:                     (time: 349)
Coordinates:
  * time                        (time) datetime64[ns] 2017-08-30 ... 2017-09-13T1...
    lon                         (time) float64 -27.64 -27.8 -27.96 ... -97.81 -97.93
    lat                         (time) float64 15.39 15.41 15.42 ... 27.41 27.49
Data variables:
    time_step                   (time) float64 1.0 1.0 1.0 1.0 ... 1.0 1.0 1.0 1.0
    radius_max_wind             (time) float64 60.0 60.0 60.0 ... 60.0 60.0 60.0
    radius_oci                  (time) float64 180.0 180.0 180.0 ... 350.0 350.0
    max_sustained_wind          (time) float64 30.0 30.67 31.33 ... 15.0 14.99 14.96
    central_pressure            (time) float64 1.008e+03 1.008e+03 ... 1.005e+03
    environmental_pressure      (time) float64 1.012e+03 1.012e+03 ... 1.008e+03
    basin                       (time) <U2 'NA' 'NA' 'NA' 'NA' ... 'NA' 'NA' 'NA'
    on_land                     (time) bool False False False ... False True True
    dist_since_lf               (time) float64 nan nan nan nan ... nan 7.605 22.71
Attributes:
    max_sustained_wind_unit:    kn
    central_pressure_unit:      mb
    name:                       IRMA_gen5
    sid:                        2017242N16333_gen5
    orig_event_flag:            False
    data_provider:              ibtracs_usa
    id_no:                      2017242016333.05
    category:                   5
```

EXERCISE

Using the first synthetic track generated,

1. Which is the time frequency of the data?
2. Compute the maximum sustained wind for each day.

```
# Put your code here
```

```
# SOLUTION:
import numpy as np
# select the track
tc_syn = tr_irma.get_track('2017242N16333_gen1')

# 1. Which is the time frequency of the data?
# The values of a DataArray are numpy.arrays.
# The nummpy.ediff1d computes the different between elements in an array
diff_time_ns = np.ediff1d(tc_syn.time)
diff_time_h = diff_time_ns.astype(int)/1000/1000/1000/60/60
print('Mean time frequency in hours:', diff_time_h.mean())
print('Std time frequency in hours:', diff_time_h.std())
print()

# 2. Compute the maximum sustained wind for each day.
print('Daily max sustained wind:', tc_syn.max_sustained_wind.groupby('time.day').
↪max())
```

```
Mean time frequency in hours: 1.0
Std time frequency in hours: 0.0

Daily max sustained wind: <xarray.DataArray 'max_sustained_wind' (day: 15)>
array([100.        , 100.        , 100.        , 123.33333333,
       155.        , 155.        , 150.        , 138.        ,
        51.85384486,  58.03963987,  29.03963987,   3.57342356,
         3.35512013,  54.        ,  99.        ])
Coordinates:
  * day      (day) int64 1 2 3 4 5 6 7 8 9 10 11 12 13 30 31
```

8.2.6 c) ECMWF Forecast Tracks

ECMWF publishes tropical cyclone forecast tracks free of charge as part of the WMO essentials. These tracks are detected automatically in the ENS and HRES models. The non-supervised nature of the model may lead to artefacts.

The `tc_fcast` trackset below inherits from TCTracks, but contains some additional metadata that follows ECMWF's definitions. Try plotting these tracks and compare them to the official cones of uncertainty! The example track at `tc_fcast.data[0]` shows the data structure.

```
# This functionality is part of climada_petals, uncomment to execute
# from climada_petals.hazard import TCForecast
#
# tc_fcast = TCForecast()
# tc_fcast.fetch_ecmwf()
#
# print(tc_fcast.data[0])
```

8.2.7 d) Load TC tracks from other sources

In addition to the *historical records of TCs (IBTrACS)*, the *probabilistic extension* of these tracks, and the *ECMWF Forecast tracks*, CLIMADA also features functions to read in synthetic TC tracks from other sources. These include synthetic storm tracks from Kerry Emanuel's coupled statistical-dynamical model (Emanuel et al., 2006 as used in Geiger et al., 2016), synthetic storm tracks from a second coupled statistical-dynamical model (CHAZ) (as described in Lee et al., 2018), and synthetic storm tracks from a fully statistical model (STORM) Bloemendaal et al., 2020). However, these functions are partly under development and/or targeted at advanced users of CLIMADA in the context of very specific use cases. They are thus not covered in this tutorial.

8.2.8 Part 2: TropCyclone() class

The `TropCyclone` class is a derived class of *Hazard*. As such, it contains all the attributes and methods of a `Hazard`. Additionally, it comes with the constructor method `from_tracks` to model tropical cyclones from tracks contained in a `TCTracks` instance.

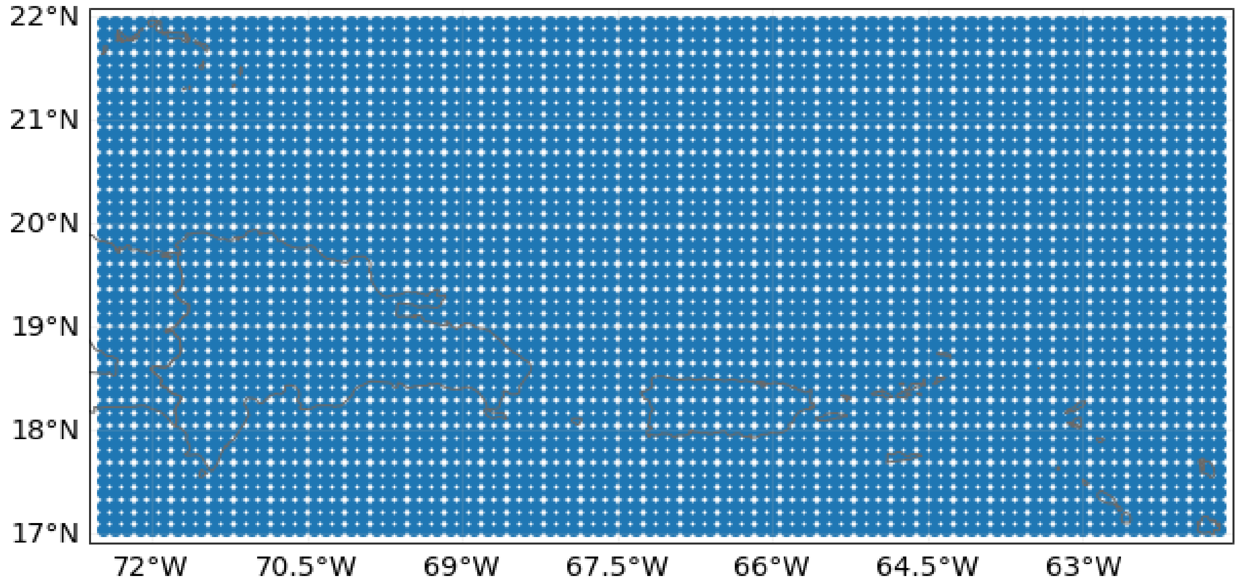
When setting tropical cyclones from tracks, the centroids where to map the wind gusts (the hazard intensity) can be provided. If no centroids are provided, the global centroids `GLB_NatID_grid_0360as_adv_2.mat` are used.

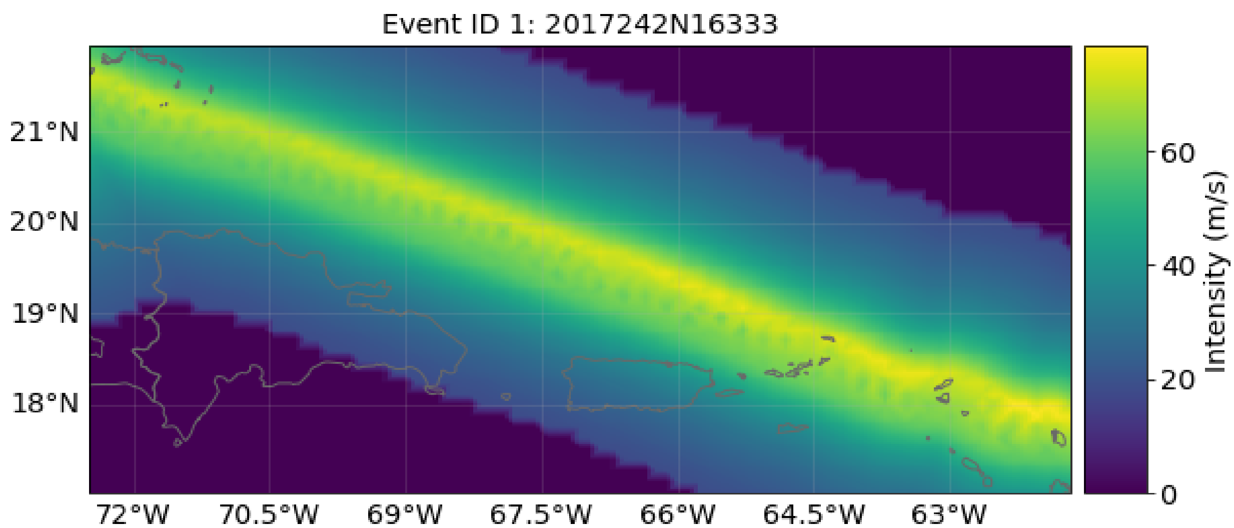
From the track properties the 1 min sustained peak gusts are computed in each centroid as the sum of a circular wind field (following Holland, 2008) and the translational wind speed that arises from the storm movement. We incorporate the decline of the translational component from the cyclone centre by multiplying it by an attenuation factor. See CLIMADA v1 and references therein for more information.

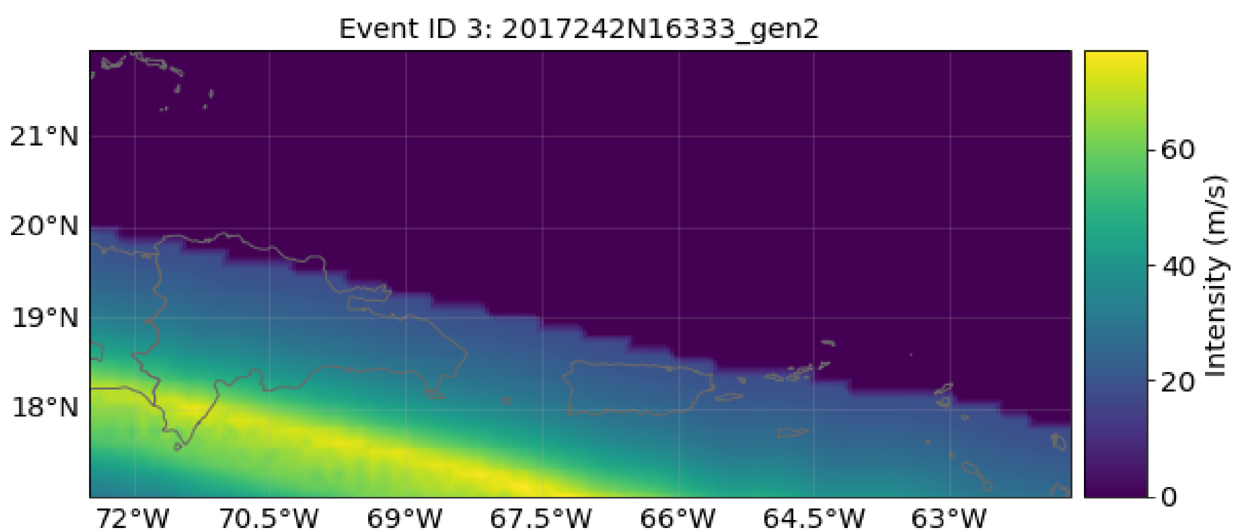
8.2.9 a) Default hazard generation for tropical cyclones

```
from climada.hazard import Centroids, TropCyclone

# construct centroids
min_lat, max_lat, min_lon, max_lon = 16.99375, 21.95625, -72.48125, -61.66875
cent = Centroids.from_pnt_bounds((min_lon, min_lat, max_lon, max_lat), res=0.12)
cent.check()
cent.plot();

# construct tropical cyclones
tc_irma = TropCyclone.from_tracks(tr_irma, centroids=cent)
# tc_irma = TropCyclone.from_tracks(tr_irma)  # try without given centroids. It might␣
↪take too much space of your memory
# and then the kernel will be killed: So, don't use this function without given␣
↪centroids!
tc_irma.check()
tc_irma.plot_intensity('2017242N16333');  # IRMA
tc_irma.plot_intensity('2017242N16333_gen2');  # IRMA's synthetic track 2
```

```
<GeoAxesSubplot:title={'center':'Event ID 3: 2017242N16333_gen2'}>
```


22°N
21°N
20°N
19°N
18°N
17°N
72°W
70.5°W
69°W
67.5°W
66°W
64.5°W
63°W


Event ID 1: 2017242N16333
21°N
20°N
19°N
18°N
72°W
70.5°W
69°W
67.5°W
66°W
64.5°W
63°W
Intensity (m/s)
0
20
40
60

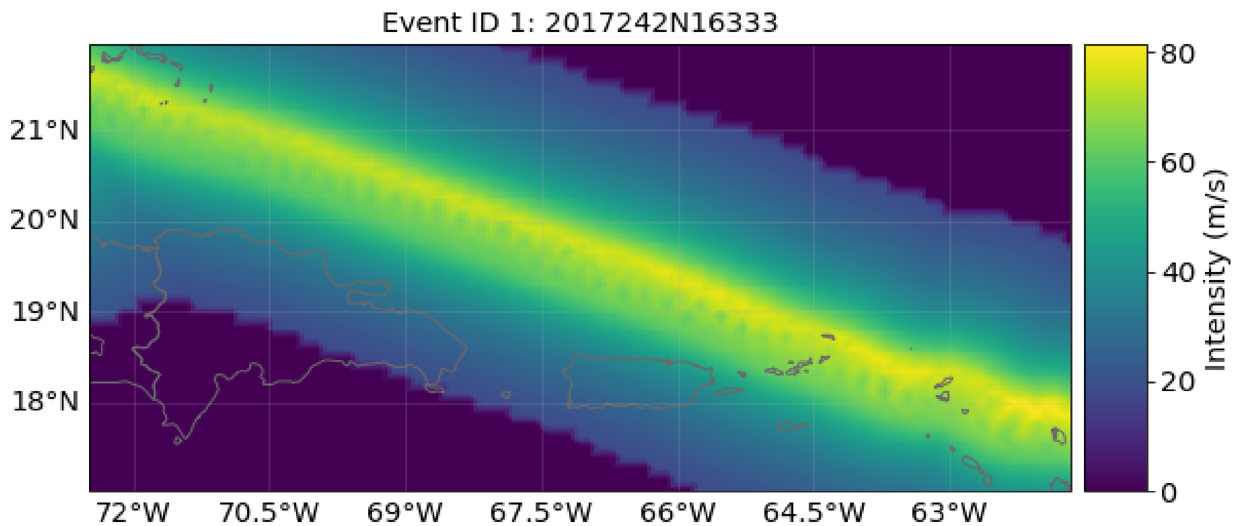

Event ID 3: 2017242N16333_gen2
21°N
20°N
19°N
18°N
72°W
70.5°W
69°W
67.5°W
66°W
64.5°W
63°W
Intensity (m/s)
0
20
40
60

8.2.10 b) Implementing climate change

`apply_climate_scenario_knu` implements the changes on intensity and frequency due to climate change described in *Global projections of intense tropical cyclone activity for the late twenty-first century from dynamical downscaling of CMIP5/RCP4.5 scenarios* of Knutson et al 2015. Other RCP scenarios are approximated from the RCP 4.5 values by interpolating them according to their relative radiative forcing.

```
# an Irma event-like in 2055 under RCP 4.5:
tc_irma = TropCyclone.from_tracks(tr_irma, centroids=cent)
tc_irma_cc = tc_irma.apply_climate_scenario_knu(ref_year=2055, rcp_scenario=45)
tc_irma_cc.plot_intensity('2017242N16333');
```

```
<GeoAxesSubplot:title={'center':'Event ID 1: 2017242N16333'}>
```



Note: this method to implement climate change is simplified and does only take into account changes in TC frequency and intensity. However, how hurricane damage changes with climate remains challenging to assess. Records of hurricane damage exhibit widely fluctuating values because they depend on rare, landfalling events which are substantially more volatile than the underlying basin-wide TC characteristics. For more accurate future projections of how a warming climate might shape TC characteristics, there is a two-step process needed. First, the understanding of how climate change affects critical environmental factors (like SST, humidity, etc.) that shape TCs is required. Second, the means of simulating how these changes impact TC characteristics (such as intensity, frequency, etc.) are necessary. Statistical-dynamical models (Emanuel et al., 2006 and Lee et al., 2018) are physics-based and allow for such climate change studies. However, this goes beyond the scope of this tutorial.

8.2.11 c) Multiprocessing - improving performance for big computations

Multiprocessing is part of the tropical cyclone module. Simply provide a process pool as method argument. Below is an example of how large amounts of data could be processed.

WARNING: Running multiprocessing code from Jupyter Notebooks can be cumbersome. It's suggested to copy the code and paste it into an interactive python console.

```
from climada.hazard import TCTracks, Centroids, TropCyclone

from pathos.pools import ProcessPool as Pool
pool = Pool() # start a pathos pool

lon_min, lat_min, lon_max, lat_max = -160, 10, -100, 36
centr = Centroids.from_pnt_bounds((lon_min, lat_min, lon_max, lat_max), 0.1)

tc_track = TCTracks.from_ibtracs_netcdf(provider='usa', year_range=(1992, 1994),␣
↪basin='EP')
tc_track.equal_timestep(pool=pool)
tc_track.calc_perturbed_trajectories(pool=pool) # OPTIONAL: if you want to generate a␣
↪probabilistic set of TC tracks.

tc_haz = TropCyclone.from_tracks(tc_track, centroids=centr, pool=pool)
tc_haz.check()

pool.close()
pool.join()
```

8.2.12 d) Making videos

Videos of a tropical cyclone hitting specific centroids can be created with the method `video_intensity()`.

WARNING: Creating an animated gif file may consume a lot of memory, up to the point where the os starts swapping or even an 'out-of-memory' exception is thrown.

```
# Note: execution of this cell will fail unless there is enough memory available (>␣
↪10G)

from climada.hazard import Centroids, TropCyclone, TCTracks

track_name = '2017242N16333' #'2016273N13300' #'1992230N11325'

tr_irma = TCTracks.from_ibtracs_netcdf(provider='usa', storm_id='2017242N16333')

lon_min, lat_min, lon_max, lat_max = -83.5, 24.4, -79.8, 29.6
centr_video = Centroids.from_pnt_bounds((lon_min, lat_min, lon_max, lat_max), 0.04)
centr_video.check()

tc_video = TropCyclone()

tc_list, tr_coord = tc_video.video_intensity(track_name, tr_irma, centr_video, file_
↪name='results/irma_tc_fl.gif')
```

```
2022-04-08 10:01:29,114 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2022-04-08 10:01:31,696 - climada.util.coordinates - INFO - dist_to_coast: UTM 32617␣
↪(1/1)
2022-04-08 10:01:38,120 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪11374 coastal centroids.
2022-04-08 10:01:38,135 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,144 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12032 coastal centroids.
2022-04-08 10:01:38,158 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,170 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,184 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,194 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,207 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,217 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,232 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,241 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,256 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,267 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,285 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,296 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,313 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,323 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,338 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,348 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,364 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,374 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,391 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,400 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,416 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,427 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,441 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,452 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,470 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,479 - climada.hazard.trop_cyclone - INFO - Generating video irma_
↪tc_fl.gif
```

```
15it [00:53,  3.60s/it]
```

`tc_list` contains a list with TropCyclone instances plotted at each time step `tr_coord` contains a list with the track path coordinates plotted at each time step

Saving disk space with mp4

Animated gif images occupy a lot of space. Using mp4 as output format makes the video sequences much smaller! However this requires the package *ffmpeg* to be installed, which is not part of the ordinary climada environment. It can be installed by executing the following command in a console:

```
conda install ffmpeg
```

Creating the same videa as above in mp4 format can be done in this way then:

```
# Note: execution of this cell will fail unless there is enough memory available (>␣
↪12G) and ffmpeg is installed

import shutil
from matplotlib import animation
from matplotlib.pyplot import rcParams

rcParams['animation.ffmpeg_path'] = shutil.which('ffmpeg')
writer = animation.FFMpegWriter(bitrate=500)
tc_list, tr_coord = tc_video.video_intensity(track_name, tr_irma, centr_video, file_
↪name='results/irma_tc_fl.mp4', writer=writer)
```

```
2022-04-08 10:03:27,161 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪11374 coastal centroids.
2022-04-08 10:03:27,182 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,192 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12032 coastal centroids.
2022-04-08 10:03:27,207 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,218 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,235 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,247 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,263 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,275 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,294 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,304 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,319 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,333 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,350 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,363 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,382 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,393 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,412 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,422 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,442 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,453 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,471 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,480 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
```

(continues on next page)

(continued from previous page)

```
2022-04-08 10:03:27,496 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,507 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,523 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,533 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,548 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,559 - climada.hazard.trop_cyclone - INFO - Generating video irma_
↪tc_fl.mp4
```

```
15it [00:55,  3.71s/it]
```

REFERENCES:

- Bloemendaal, N., Haigh, I. D., de Moel, H., Muis, S., Haarsma, R. J., & Aerts, J. C. J. H. (2020). Generation of a global synthetic tropical cyclone hazard dataset using STORM. Scientific Data, 7(1). https://doi.org/10.1038/s41597-020-0381-2
- Emanuel, K., S. Ravela, E. Vivant, and C. Risi, 2006: A Statistical Deterministic Approach to Hurricane Risk Assessment. Bull. Amer. Meteor. Soc., 87, 299–314, https://doi.org/10.1175/BAMS-87-3-299.
- Geiger, T., Frieler, K., & Levermann, A. (2016). High-income does not protect against hurricane losses. Environmental Research Letters, 11(8). https://doi.org/10.1088/1748-9326/11/8/084012
- Knutson, T. R., Sirutis, J. J., Zhao, M., Tuleya, R. E., Bender, M., Vecchi, G. A., … Chavas, D. (2015). Global projections of intense tropical cyclone activity for the late twenty-first century from dynamical downscaling of CMIP5/RCP4.5 scenarios. Journal of Climate, 28(18), 7203–7224. https://doi.org/10.1175/JCLI-D-15-0129.1
- Lee, C. Y., Tippett, M. K., Sobel, A. H., & Camargo, S. J. (2018). An environmentally forced tropical cyclone hazard model. Journal of Advances in Modeling Earth Systems, 10(1), 223–241. https://doi.org/10.1002/2017MS001186

8.3 Hazard: winter windstorms / extratropical cyclones in Europe

8.3.1 Or: The StormEurope hazard subclass of CLIMADA

Auth: Jan Hartman & Thomas Röösli
Date: 2018-04-26 & 2020-03-03

This notebook will give a quick tour of the capabilities of the StormEurope hazard class. This includes functionalities to apply probabilistic alterations to historical storms.

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [15, 10]
```

8.3.2 Reading Data

StormEurope was written under the presumption that you'd start out with WISC storm footprint data in netCDF format. This notebook works with a demo dataset. If you would like to work with the real data: (1) Please follow the link and download the file C3S_WISC_FOOTPRINT_NETCDF_0100.tgz from the Copernicus Windstorm Information Service, (2) unzip it (3) uncomment the last two lines in the following codeblock and (4) adjust the variable "WISC_files".

We first construct an instance and then point the reader at a directory containing compatible `.nc` files. Since there are other files in there, we must be explicit and use a globbing pattern; supplying incompatible files will make the reader fail.

The reader actually calls `climada.util.files_handler.get_file_names`, so it's also possible to hand it an explicit list of filenames, or a dirname, or even a list of glob patterns or directories.

```
from climada.hazard import StormEurope
from climada.util.constants import WS_DEMO_NC

storm_instance = StormEurope.from_footprints(WS_DEMO_NC)

# WISC_files = '/path/to/folder/C3S_WISC_FOOTPRINT_NETCDF_0100/fp_era[!er5]*_0.nc'
# storm_instance = StormEurope.from_footprints(WISC_files)
```

```
$CLIMADA_SRC/climada/hazard/centroids/centr.py:822: UserWarning: Geometry is in a␣
↪geographic CRS. Results from 'buffer' are likely incorrect. Use 'GeoSeries.to_crs()
↪' to re-project geometries to a projected CRS before this operation.

  xy_pixels = self.geometry.buffer(res / 2).envelope
```

8.3.3 Introspection

Let's quickly see what attributes this class brings with it:

```
storm_instance?
```

```
Type:           StormEurope
String form:    <climada.hazard.storm_europe.StormEurope object at 0x7f2a986b4c70>
File:           ~/code/climada_python/climada/hazard/storm_europe.py
Docstring:
A hazard set containing european winter storm events. Historic storm
events can be downloaded at http://wisc.climate.copernicus.eu/ and read
with `from_footprints`. Weather forecasts can be automatically downloaded from
https://opendata.dwd.de/ and read with from_icon_grib(). Weather forecast
from the COSMO-Consortium http://www.cosmo-model.org/ can be read with
from_cosmoe_file().

Attributes
----------
ssi_wisc : np.array, float
    Storm Severity Index (SSI) as recorded in
    the footprint files; apparently not reproducible from the footprint
    values only.
ssi : np.array, float
    SSI as set by set_ssi; uses the Dawkins
    definition by default.
Init docstring: Calls the Hazard init dunder. Sets unit to 'm/s'.
```

You could also try listing all permissible methods with dir(storm_instance), but since that would include the methods from the Hazard base class, you wouldn't know what's special. The best way is to read the source: uncomment the following statement to read more.

```
# StormEurope??
```

8.3.4 Into the Storm Severity Index (SSI)

The SSI, according to Dawkins et al. 2016 or Lamb and Frydendahl, 1991, can be set using set_ssi. For demonstration purposes, I show the default arguments. (Check also the defaults using storm_instance.calc_ssi?, the method for which set_ssi is a wrapper.)

We won't be using the plot_ssi functionality just yet, because we only have two events; the graph really isn't informative. After this, we'll generate some more storms to make that plot more aesthetically pleasing.

```
storm_instance.set_ssi(
    method = 'wind_gust',
    intensity = storm_instance.intensity,
    # the above is just a more explicit way of passing the default
    on_land = True,
    threshold = 25,
    sel_cen = None
    # None is default. sel_cen could be used to subset centroids
)
```

8.3.5 Probabilistic Storms

This class allows generating probabilistic storms from historical ones according to a method outlined in Schwierz et al. 2010. This means that per historical event, we generate 29 new ones with altered intensities. Since it's just a bunch of vector operations, this is pretty fast.

However, we should not return the entire probabilistic dataset in-memory: in trials, this used up 60 GB of RAM, thus requiring a great amount of swap space. Instead, we must select a country by setting the reg_id parameter to an ISO_N3 country code used in the Natural Earth dataset. It is also possible to supply a list of ISO codes. If your machine is up for the job of handling the whole dataset, set the reg_id parameter to None.

Since assigning each centroid a country ID is a rather inefficient affair, you may need to wait a minute or two for the entire WISC dataset to be processed. For the small demo dataset, it runs pretty quickly.

```
%%time
storm_prob = storm_instance.generate_prob_storms(reg_id=528)
storm_prob.plot_intensity(0);
```

```
2020-03-05 10:29:31,845 - climada.hazard.centroids.centr - INFO - Setting geometry␣
↪points.
2020-03-05 10:29:32,248 - climada.hazard.centroids.centr - DEBUG - Setting region_id␣
↪9944 points.
2020-03-05 10:29:32,466 - climada.util.coordinates - DEBUG - Setting region_id 9944␣
↪points.
2020-03-05 10:29:33,506 - climada.hazard.storm_europe - INFO - Commencing␣
↪probabilistic calculations
2020-03-05 10:29:33,620 - climada.hazard.storm_europe - INFO - Generating new␣
↪StormEurope instance
2020-03-05 10:29:33,663 - climada.util.checker - DEBUG - Hazard.ssi not set.
```

(continues on next page)

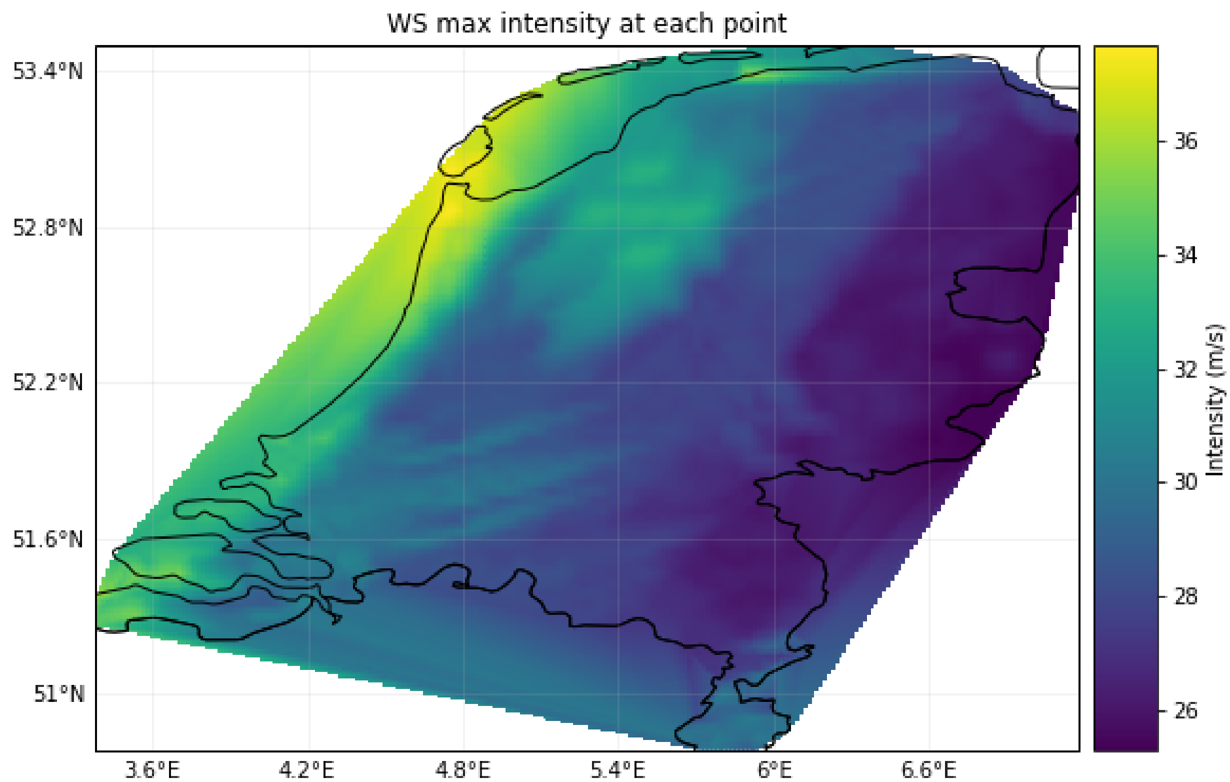
(continued from previous page)

```
2020-03-05 10:29:33,664 - climada.util.checker - DEBUG - Hazard.ssi_wisc not set.
2020-03-05 10:29:33,665 - climada.util.checker - DEBUG - Hazard.event_name not set.␣
↪Default values set.
```

```
C:\shortpaths\GitHub\climada_python\climada\util\plot.py:311: UserWarning: Tight␣
↪layout not applied. The left and right margins cannot be made large enough to␣
↪accommodate all axes decorations.
  fig.tight_layout()
```

```
Wall time: 2.24 s
```

```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x1dafba69940>
```

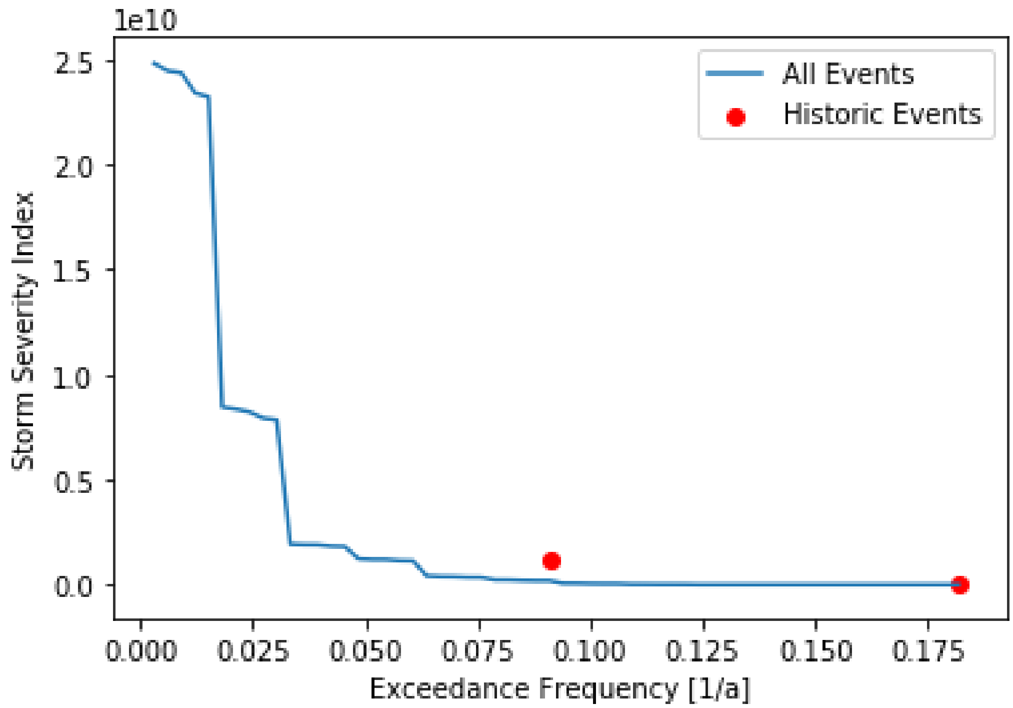


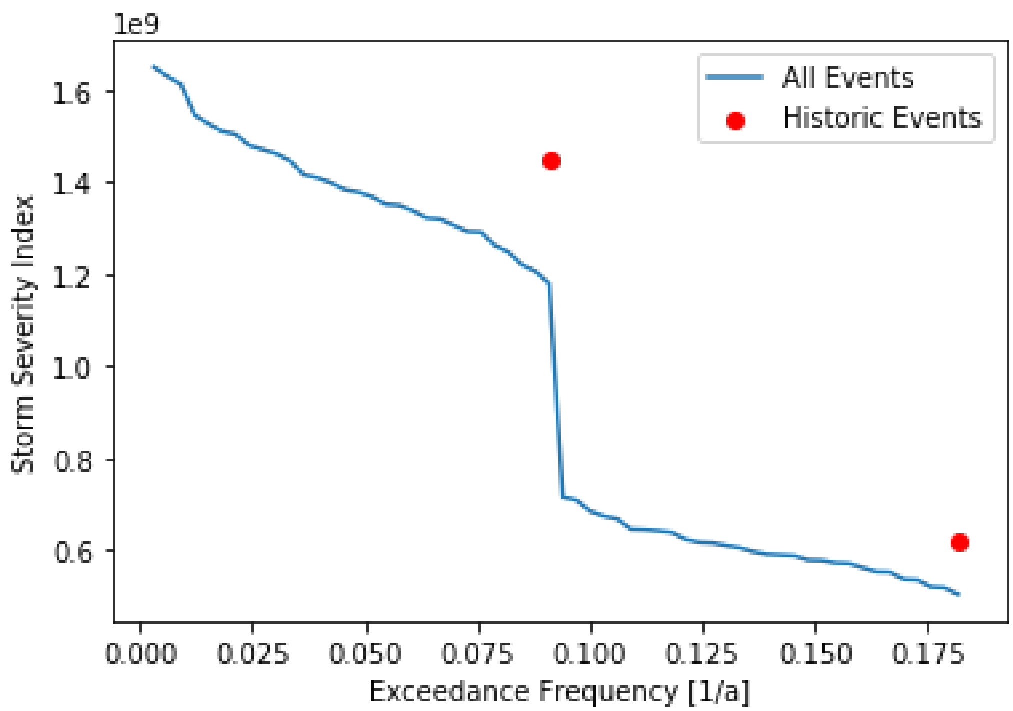
We can get much more fancy in our calls to `generate_prob_storms`; the keyword arguments after `ssi_args` are passed on to `_hist2prob`, allowing us to tweak the probabilistic permutations.

```
ssi_args = {
    'on_land': True,
    'threshold': 25,
}

storm_prob_xtreme = storm_instance.generate_prob_storms(
    reg_id=[56, 528], # BEL and NLD
    spatial_shift=2,
    ssi_args=ssi_args,
    power=1.5,
    scale=0.3
)
```

We can now check out the SSI plots of both these calculations. The comparison between the historic and probabilistic ssi values, only makes sense for the full dataset.

```
storm_prob_xtreme.plot_ssi(full_area=True);
storm_prob.plot_ssi(full_area=True);
```





```
(<Figure size 1080x720 with 1 Axes>,
 <AxesSubplot:xlabel='Exceedance Frequency [1/a]', ylabel='Storm Severity Index'>)
```

CHAPTER

NINE

EXPOSURES TUTORIALS

9.1 Exposures class

9.1.1 Content

- *Defining exposuresfrom your owndata*
- *Loading CLIMADA generated exposure files or generating new ones*
- *Visualize Exposures*
- *Write (Save) Exposures*
- *Dask: improving performance for big exposure*

What is an exposure?

Exposure describes the set of assets, people, livelihoods, infrastructures, etc. within an area of interest in terms of their geographic location, their value etc.; in brief - everything potentially exposed to hazards.

What options does CLIMADA offer for me to create an exposure?

CLIMADA has an `Exposures` class for this purpuse. An `Exposures` instance can be filled with your own data, or loaded from available default sources implemented through some Exposures-type classes from CLIMADA. If you have your own data, they can be provided in the formats of a `pandas.DataFrame`, a `geopandas.GeoDataFrame` or simply an `Excel` file. If you didn't collect your own data, exposures can be generated on the fly using CLIMADA's *LitPop*, BlackMarble or OpenStreetMap modules. See the respective tutorials to learn what exactly they contain and how to use them.

What does an exposure look like in CLIMADA?

An exposure is represented in the class `Exposures`, which contains a geopandas GeoDataFrame that is accessible through the `Exposures` `gdf` attribute. Certain columns of `gdf` *have to* be specified, while others are optional (this means that the package `climada.engine` also works without these variables set.) The full list looks like this:

Mandatory columns	Data Type	Description
`latitude`	float	latitude
`longitude`	float	longitude
`value`	float	a value for each exposure

Optional column	Data Type	Description
impf_	int	impact functions ids for hazard types.important attribute, since it relates the exposures to the hazard by specifying the impf_act functions.Ideally it should be set to the specific hazard (e.g. `impf_TC`) so that different hazards can be setin the same Exposures (e.g. `impf_TC` and `impf_FL`).If not provided, set to default `impf_` with ids 1 in check().
geometry	Point	geometry of type PointMain feature of geopandas DataFrame extensionComputed in method `set_geometry_points()`
deducti	float	deductible value for each exposure. Used for insurance
cover	float	cover value for each exposure. Used for insurance
category_	int	category id (e.g. building code) for each exposure
region_	int	region id (e.g. country ISO code) for each exposure
impf_	int	centroids index for hazard type.There might be different hazards defined: centr_TC, centr_FL, …Computed in method `assign_centroids()`

Metadata variables	Data Type	Description
crs	str or int	coordinate reference system, see GeoDataFrame.crs
description	str	describing origin and content of the exposures data
ref_yea	int	reference year
value_u	str	unit of the exposures' values
meta	dict	dictionary containing corresponding raster properties (if any):width, height, crs and transform must be present at least (transform needs to contain upper left corner!).Exposures might not contain all the points of the corresponding raster.

9.1.2 Defining exposures from your own data

The essential structure of an exposure is similar, irrespective of the data type you choose to provide: As mentioned in the introduction, the key variables to be provided are `latitudes`, `longitudes` and `values` of your exposed assets. While not mandatory, but very useful to provide for the impact calculation at later stages: the impact function id (see `impf_*` in the table above). The following examples will walk you through how to specifiy those four variables, and demonstrate the use of a few more optional parameters on the go.

Exposures from a pandas DataFrame

In case you are unfamiliar with the data structure, check out the pandas DataFrame documentation.

```
import numpy as np
from pandas import DataFrame
from climada.entity import Exposures

# Fill a pandas DataFrame with the 3 mandatory variables (latitude, longitude, value)␣
↪for a number of assets (10'000).
# We will do this with random dummy data for purely illustrative reasons:
exp_df = DataFrame()
n_exp = 100*100
# provide value
exp_df['value'] = np.arange(n_exp)
# provide latitude and longitude
lat, lon = np.mgrid[15 : 35 : complex(0, np.sqrt(n_exp)), 20 : 40 : complex(0, np.
↪sqrt(n_exp))]
exp_df['latitude'] = lat.flatten()
exp_df['longitude'] = lon.flatten()
```

```
# For each exposure entry, specify which impact function should be taken for which␣
↪hazard type.
# In this case, we only specify the IDs for tropical cyclone (TC); here, each␣
↪exposure entry will be treated with
# the same impact function: the one that has ID '1':
# Of course, this will only be relevant at later steps during impact calculations.
exp_df['impf_TC'] = np.ones(n_exp, int)
```

```
# Let's have a look at the pandas DataFrame
print('\x1b[1;03;30;30m' + 'exp_df is a DataFrame:', str(type(exp_df)) + '\x1b[0m')
print('\x1b[1;03;30;30m' + 'exp_df looks like:' + '\x1b[0m')
print(exp_df.head())
```

```
exp_df is a DataFrame: <class 'pandas.core.frame.DataFrame'>
exp_df looks like:
   value  latitude  longitude  impf_TC
0      0      15.0  20.000000        1
1      1      15.0  20.202020        1
2      2      15.0  20.404040        1
3      3      15.0  20.606061        1
4      4      15.0  20.808081        1
```

```
# Generate Exposures from the pandas DataFrame. This step converts the DataFrame into
# a CLIMADA Exposures instance!
exp = Exposures(exp_df)
print('\n\x1b[1;03;30;30m' + 'exp has the type:', str(type(exp)))
print('and contains a GeoDataFrame exp.gdf:', str(type(exp.gdf)) + '\n\n\x1b[0m')

# set geometry attribute (shapely Points) from GeoDataFrame from latitude and␣
↪longitude
exp.set_geometry_points()
print('\n' + '\x1b[1;03;30;30m' + 'check method logs:' + '\x1b[0m')

# always apply the check() method in the end. It puts metadata that has not been␣
↪assigned,
```

(continues on next page)

(continued from previous page)

```
# and points out missing mandatory data
exp.check()
```

```
exp has the type: <class 'climada.entity.exposures.base.Exposures'>
and contains a GeoDataFrame exp.gdf: <class 'geopandas.geodataframe.GeoDataFrame'>



check method logs:
```

```
# let's have a look at the Exposures instance we created!
print('\n' + '\x1b[1;03;30;30m'  + 'exp looks like:' + '\x1b[0m')
print(exp)
```

```
exp looks like:
ref_year: 2018
value_unit: USD
meta: {'crs': 'EPSG:4326'}
crs: EPSG:4326
data:
      value  latitude  longitude  impf_TC                  geometry
0         0      15.0  20.000000        1  POINT (20.00000 15.00000)
1         1      15.0  20.202020        1  POINT (20.20202 15.00000)
2         2      15.0  20.404040        1  POINT (20.40404 15.00000)
3         3      15.0  20.606061        1  POINT (20.60606 15.00000)
4         4      15.0  20.808081        1  POINT (20.80808 15.00000)
...     ...       ...        ...      ...                        ...
9995   9995      35.0  39.191919        1  POINT (39.19192 35.00000)
9996   9996      35.0  39.393939        1  POINT (39.39394 35.00000)
9997   9997      35.0  39.595960        1  POINT (39.59596 35.00000)
9998   9998      35.0  39.797980        1  POINT (39.79798 35.00000)
9999   9999      35.0  40.000000        1  POINT (40.00000 35.00000)

[10000 rows x 5 columns]
```

Exposures from a geopandas GeoDataFrame

In case you are unfamiliar with with data structure, check out the geopandas GeoDataFrame documentation. The main difference to the example above (pandas DataFrame) is that, while previously, we provided latitudes and longitudes which were then converted to a `geometry` GeoSeries using the `set_geometry_points` method, GeoDataFrames alread come with a defined `geometry` GeoSeries. In this case, we take the geometry info and use the `set_lat_lon` method to explicitly provide latitudes and longitudes. This example focuses on data with POINT geometry, but in principle, other geometry types (such as POLYGON and MULTIPOLYGON) would work as well.

```
import numpy as np
import geopandas as gpd
from climada.entity import Exposures

# Read spatial info from an external file into GeoDataFrame
world = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))
print('\x1b[1;03;30;30m' + 'World is a GeoDataFrame:', str(type(world)) + '\x1b[0m')
print('\x1b[1;03;30;30m' + 'World looks like:' + '\x1b[0m')
print(world.head())
```

```
World is a GeoDataFrame: <class 'geopandas.geodataframe.GeoDataFrame'>
World looks like:
           name                    geometry
0  Vatican City  POINT (12.45339 41.90328)
1    San Marino  POINT (12.44177 43.93610)
2         Vaduz   POINT (9.51667 47.13372)
3    Luxembourg   POINT (6.13000 49.61166)
4       Palikir  POINT (158.14997 6.91664)
```

```
# Generate Exposures: value, latitude and longitude for each exposure entry.
# Convert GeoDataFrame into Exposure instance
exp_gpd = Exposures(world)
print('\n' + '\x1b[1;03;30;30m' + 'exp_gpd is an Exposures:', str(type(exp_gpd)) + '\
↪x1b[0m')
# add random values to entries
exp_gpd.gdf['value'] = np.arange(world.shape[0])
# set latitude and longitude attributes from geometry
exp_gpd.set_lat_lon()
```

```
exp_gpd is an Exposures: <class 'climada.entity.exposures.base.Exposures'>
```

```
# For each exposure entry, specify which impact function should be taken for which␣
↪hazard type.
# In this case, we only specify the IDs for tropical cyclone (TC); here, each␣
↪exposure entry will be treated with
# the same impact function: the one that has ID '1':
# Of course, this will only be relevant at later steps during impact calculations.
exp_gpd.gdf['impf_TC'] = np.ones(world.shape[0], int)
print('\n' + '\x1b[1;03;30;30m' + 'check method logs:' + '\x1b[0m')

# as always, run check method to assign meta-data and check for missing mandatory␣
↪variables.
exp_gpd.check()
```

```
check method logs:
```

```
# let's have a look at the Exposures instance we created!
print('\n' + '\x1b[1;03;30;30m'  + 'exp_gpd looks like:' + '\x1b[0m')
print(exp_gpd)
```

```
exp_gpd looks like:
ref_year: 2018
value_unit: USD
meta: {'crs': <Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
}
```

(continues on next page)

(continued from previous page)

```
crs: epsg:4326
data:
             name                    geometry  value   latitude   longitude  \
0    Vatican City    POINT (12.45339 41.90328)      0  41.903282   12.453387
1      San Marino    POINT (12.44177 43.93610)      1  43.936096   12.441770
2           Vaduz     POINT (9.51667 47.13372)      2  47.133724    9.516669
3      Luxembourg     POINT (6.13000 49.61166)      3  49.611660    6.130003
4         Palikir    POINT (158.14997 6.91664)      4   6.916644  158.149974
..            ...                          ...    ...        ...         ...
197         Cairo    POINT (31.24802 30.05191)    197  30.051906   31.248022
198         Tokyo   POINT (139.74946 35.68696)    198  35.686963  139.749462
199         Paris     POINT (2.33139 48.86864)    199  48.868639    2.331389
200      Santiago  POINT (-70.66899 -33.44807)    200 -33.448068  -70.668987
201     Singapore    POINT (103.85387 1.29498)    201   1.294979  103.853875

     impf_TC
0          1
1          1
2          1
3          1
4          1
..       ...
197        1
198        1
199        1
200        1
201        1

[202 rows x 6 columns]
```

The fact that Exposures is built around a geopandas.GeoDataFrame offers all the useful functionalities that come with the package. The following examples showcase only a few of those.

```
# Example 1: extract data in a region: latitudes between -5 and 5
sel_exp = exp_gpd.copy()  # to keep the original exp_gpd Exposures data
sel_exp.gdf = sel_exp.gdf.cx[:, -5:5]

print('\n' + '\x1b[1;03;30;30m' + 'sel_exp contains a subset of the original data' +
↪'\x1b[0m')
sel_exp.gdf.head()
```

```
sel_exp contains a subset of the original data
```

```
         name                   geometry  value  latitude   longitude  impf_TC
9      Tarawa  POINT (173.01757 1.33819)      9  1.338188  173.017571        1
13     Kigali  POINT (30.05859 -1.95164)     13 -1.951644   30.058586        1
15       Juba   POINT (31.58003 4.82998)     15  4.829975   31.580026        1
27  Bujumbura  POINT (29.36001 -3.37609)     27 -3.376087   29.360006        1
48    Kampala   POINT (32.58138 0.31860)     48  0.318605   32.581378        1
```

```
# Example 2: extract data in a polygon
from shapely.geometry import Polygon
sel_polygon = exp_gpd.copy()  # to keep the original exp_gpd Exposures data

poly = Polygon([(0, -10), (0, 10), (10, 5)])
```

(continues on next page)

(continued from previous page)

```
sel_polygon.gdf = sel_polygon.gdf[sel_polygon.gdf.intersects(poly)]

# Let's have a look. Again, the sub-selection is a GeoDataFrame!
print('\n' + '\x1b[1;03;30;30m' + 'sel_exp contains a subset of the original data' +
↪'\x1b[0m')
sel_polygon.gdf
```

```
sel_exp contains a subset of the original data
```

```
         name                  geometry  value  latitude  longitude  impf_TC
36       Lome  POINT (1.22081 6.13388)     36  6.133883   1.220811        1
84     Malabo  POINT (8.78328 3.75002)     84  3.750015   8.783278        1
113   Cotonou  POINT (2.51804 6.40195)    113  6.401954   2.518045        1
125  Sao Tome  POINT (6.73333 0.33340)    125  0.333402   6.733325        1
```

```
# Example 3: change coordinate reference system
# use help to see more options: help(sel_exp.to_crs)
sel_polygon.to_crs(epsg=3395, inplace=True)
print('\n' + '\x1b[1;03;30;30m' + 'the crs has changed to ' +str(sel_polygon.crs))
print('the values for latitude and longitude are now according to the new coordinate␣
↪system: \x1b[0m')
sel_polygon.gdf
```

```
the crs has changed to epsg:3395
the values for latitude and longitude are now according to the new coordinate system:
```

```
         name                        geometry  value       latitude  \
36       Lome  POINT (135900.088 679566.334)     36  679566.333952
84     Malabo  POINT (977749.984 414955.551)     84  414955.550857
113   Cotonou  POINT (280307.458 709388.810)    113  709388.810160
125  Sao Tome   POINT (749550.327 36865.909)    125   36865.908682

         longitude  impf_TC
36   135900.087901        1
84   977749.983897        1
113  280307.458315        1
125  749550.327404        1
```

```
# Example 4: concatenate exposures
exp_all = Exposures.concat([sel_polygon, sel_exp.to_crs(epsg=3395)])

# the output is of type Exposures
print('exp_all type and number of rows:', type(exp_all), exp_all.gdf.shape[0])
print('number of unique rows:', exp_all.gdf.drop_duplicates().shape[0])

# NaNs will appear in the missing values
exp_all.gdf.head()
```

```
exp_all type and number of rows: <class 'climada.entity.exposures.base.Exposures'> 25
number of unique rows: 23
```

```
       name                        geometry  value       latitude  \
0      Lome    POINT (135900.088 679566.334)     36  679566.333952
1    Malabo    POINT (977749.984 414955.551)     84  414955.550857
```

(continues on next page)

(continued from previous page)

```
2   Cotonou    POINT (280307.458 709388.810)    113  709388.810160
3  Sao Tome     POINT (749550.327 36865.909)    125   36865.908682
4    Tarawa  POINT (19260227.883 147982.749)      9  147982.748978

      longitude  impf_TC
0  1.359001e+05        1
1  9.777500e+05        1
2  2.803075e+05        1
3  7.495503e+05        1
4  1.926023e+07        1
```

Exposures of any file type supported by Geopandas and Pandas

Geopandas can read almost any vector-based spatial data format including ESRI shapefile, GeoJSON files and more, see readers geopandas. Pandas supports formats such as csv, html or sql; see readers pandas. Using the corresponding readers, `DataFrame` and `GeoDataFrame` can be filled and provided to `Exposures` following the previous examples.

Exposures from an excel file

If you manually collect exposure data, Excel may be your preferred option. In this case, it is easiest if you format your data according to the structure provided in the template `climada_python/climada/data/system/entity_template.xlsx`, in the sheet `assets`.

```
import pandas as pd
from climada.util.constants import ENT_TEMPLATE_XLS
from climada.entity import Exposures

# Read your Excel file into a pandas DataFrame (we will use the template example for␣
↪this demonstration):
file_name = ENT_TEMPLATE_XLS
exp_templ = pd.read_excel(file_name)

# Let's have a look at the data:
print('\x1b[1;03;30;30m' + 'exp_templ is a DataFrame:', str(type(exp_templ)) + '\
↪x1b[0m')
print('\x1b[1;03;30;30m' + 'exp_templ looks like:' + '\x1b[0m')
exp_templ.head()
```

```
exp_templ is a DataFrame: <class 'pandas.core.frame.DataFrame'>
exp_templ looks like:
```

```
    latitude  longitude         value  deductible         cover  region_id  \
0  26.933899 -80.128799  1.392750e+10           0  1.392750e+10          1
1  26.957203 -80.098284  1.259606e+10           0  1.259606e+10          1
2  26.783846 -80.748947  1.259606e+10           0  1.259606e+10          1
3  26.645524 -80.550704  1.259606e+10           0  1.259606e+10          1
4  26.897796 -80.596929  1.259606e+10           0  1.259606e+10          1

   category_id  impf_TC  centr_TC  impf_FL  centr_FL
0            1        1         1        1         1
1            1        1         2        1         2
2            1        1         3        1         3
```

(continues on next page)

(continued from previous page)

```
3           1         1         4         1         4
4           1         1         5         1         5
```

As we can see, the general structure is the same as always: the exposure has `latitude`, `longitude` and `value` columns. Further, this example specified several impact function ids: some for Tropical Cyclones (`impf_TC`), and some for Floods (`impf_FL`). It also provides some meta-info (`region_id`, `category_id`) and insurance info relevant to the impact calculation in later steps (`cover`, `deductible`).

```
# Generate an Exposures instance from the dataframe.
exp_templ = Exposures(exp_templ)
print('\n' + '\x1b[1;03;30;30m' + 'exp_templ is now an Exposures:', str(type(exp_
↪templ)) + '\x1b[0m')

# set geometry attribute (shapely Points) from GeoDataFrame from latitude and␣
↪longitude
print('\n' + '\x1b[1;03;30;30m' + 'set_geometry logs:' + '\x1b[0m')
exp_templ.set_geometry_points()
# as always, run check method to include metadata and check for missing mandatory␣
↪parameters

print('\n' + '\x1b[1;03;30;30m' + 'check exp_templ:' + '\x1b[0m')
exp_templ.check()
```

```
exp_templ is now an Exposures: <class 'climada.entity.exposures.base.Exposures'>

set_geometry logs:

check exp_templ:
```

```
# Let's have a look at our Exposures instance!
print('\n' + '\x1b[1;03;30;30m'  + 'exp_templ.gdf looks like:' + '\x1b[0m')
exp_templ.gdf.head()
```

```
exp_templ.gdf looks like:
```

```
    latitude  longitude         value  deductible         cover  region_id  \
0  26.933899 -80.128799  1.392750e+10           0  1.392750e+10          1
1  26.957203 -80.098284  1.259606e+10           0  1.259606e+10          1
2  26.783846 -80.748947  1.259606e+10           0  1.259606e+10          1
3  26.645524 -80.550704  1.259606e+10           0  1.259606e+10          1
4  26.897796 -80.596929  1.259606e+10           0  1.259606e+10          1

   category_id  impf_TC  centr_TC  impf_FL  centr_FL  \
0            1        1         1        1         1
1            1        1         2        1         2
2            1        1         3        1         3
3            1        1         4        1         4
4            1        1         5        1         5

                    geometry
0  POINT (-80.12880 26.93390)
1  POINT (-80.09828 26.95720)
2  POINT (-80.74895 26.78385)
3  POINT (-80.55070 26.64552)
4  POINT (-80.59693 26.89780)
```

Exposures from a raster file

Last but not least, you may have your exposure data stored in a raster file. Raster data may be read in from any file-type supported by rasterio.

```
from rasterio.windows import Window
from climada.util.constants import HAZ_DEMO_FL
from climada.entity import Exposures

# We take an example with a dummy raster file (HAZ_DEMO_FL), running the method set_
↪from_raster directly loads the
# necessary info from the file into an Exposures instance.
exp_raster = Exposures.from_raster(HAZ_DEMO_FL, window= Window(10, 20, 50, 60))
# There are several keyword argument options that come with the set_from_raster␣
↪method (such as
# specifying a window, if not the entire file should be read, or a bounding box.␣
↪Check them out.
```

```
# As always, run the check method, such that metadata can be assigned and checked for␣
↪missing mandatory parameters.
exp_raster.check()
print('Meta:', exp_raster.meta)
```

```
Meta: {'driver': 'GSBG', 'dtype': 'float32', 'nodata': 1.701410009187828e+38, 'width
↪': 50, 'height': 60, 'count': 1, 'crs': CRS.from_epsg(4326), 'transform': Affine(0.
↪009000000000000341, 0.0, -69.2471495969998,
       0.0, -0.009000000000000341, 10.248220966978932)}
```

```
# Let's have a look at the Exposures instance!
print('\n' + '\x1b[1;03;30;30m'  + 'exp_raster looks like:' + '\x1b[0m')
exp_raster.gdf.head()
```

```
exp_raster looks like:
```

	longitude	latitude	value	impf_
0	-69.24265	10.243721	0.0	1
1	-69.23365	10.243721	0.0	1
2	-69.22465	10.243721	0.0	1
3	-69.21565	10.243721	0.0	1
4	-69.20665	10.243721	0.0	1

9.1.3 Loading CLIMADA-generated exposure files or generating new ones

In case you already have a CLIMADA-generated file containing `Exposures` info, you can of course load it back into memory. Most likely, the data format will either be of `.hdf5` or of `.mat`.

In case you neither have your own data, nor a CLIMADA-generated file, you can also create an exposure on the fly using one of the three CLIMADA-internal exposure generators: *LitPop*, BlackMarble or OpenStreetMap modules. The latter three are extensively described in their own, linked, tutorials.

```
# read  generated with the Python version with from_hdf5()
# note: for .mat data, use the method from_mat() analogously.
from climada.util.constants import EXP_DEMO_H5
```

(continues on next page)

(continued from previous page)

```
exp_hdf5 = Exposures.from_hdf5(EXP_DEMO_H5)
exp_hdf5.check()
print(type(exp_hdf5))
```

```
<class 'climada.entity.exposures.base.Exposures'>
```

Before you leave ...

After defining an `Exposures` instance use always the `check()` method to see which attributes are missing. This method will raise an ERROR if `value`, `longitude` or `latitude` ar missing and an INFO messages for the the optional variables not set.

9.1.4 Visualize Exposures

The method `plot_hexbin()` uses cartopy and matplotlib's hexbin function to represent the exposures values as 2d bins over a map. Configure your plot by fixing the different inputs of the method or by modifying the returned `matplotlib` figure and axes.

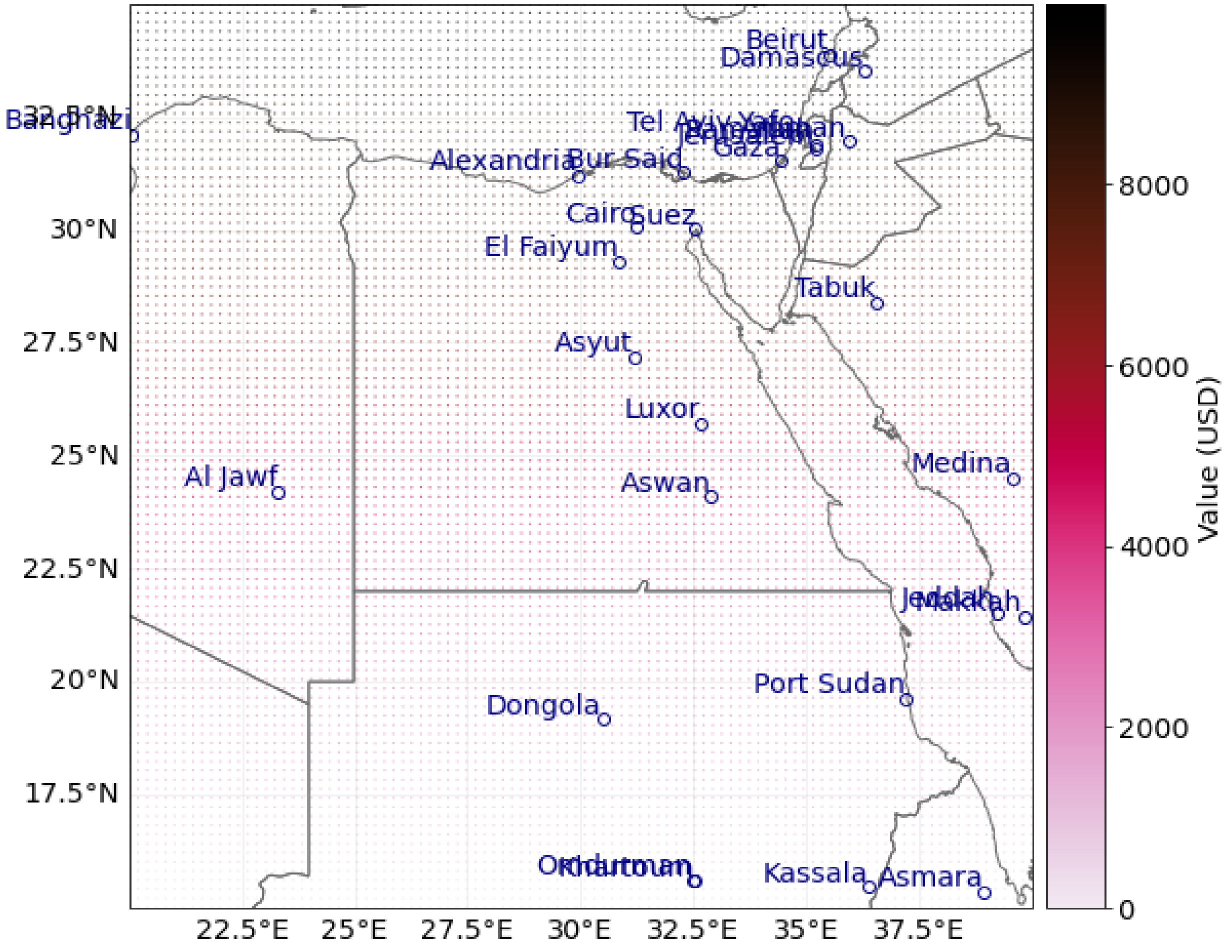
The method `plot_scatter()` uses cartopy and matplotlib's scatter function to represent the points values over a 2d map. As usal, it returns the figure and axes, which can be modify aftwerwards.

The method `plot_raster()` rasterizes the points into the given resolution. Use the `save_tiff` option to save the resulting `tiff` file and the `res_raster`option to re-set the raster's resolution.

Finally, the method `plot_basemap()` plots the scatter points over a satellite image using contextily library.

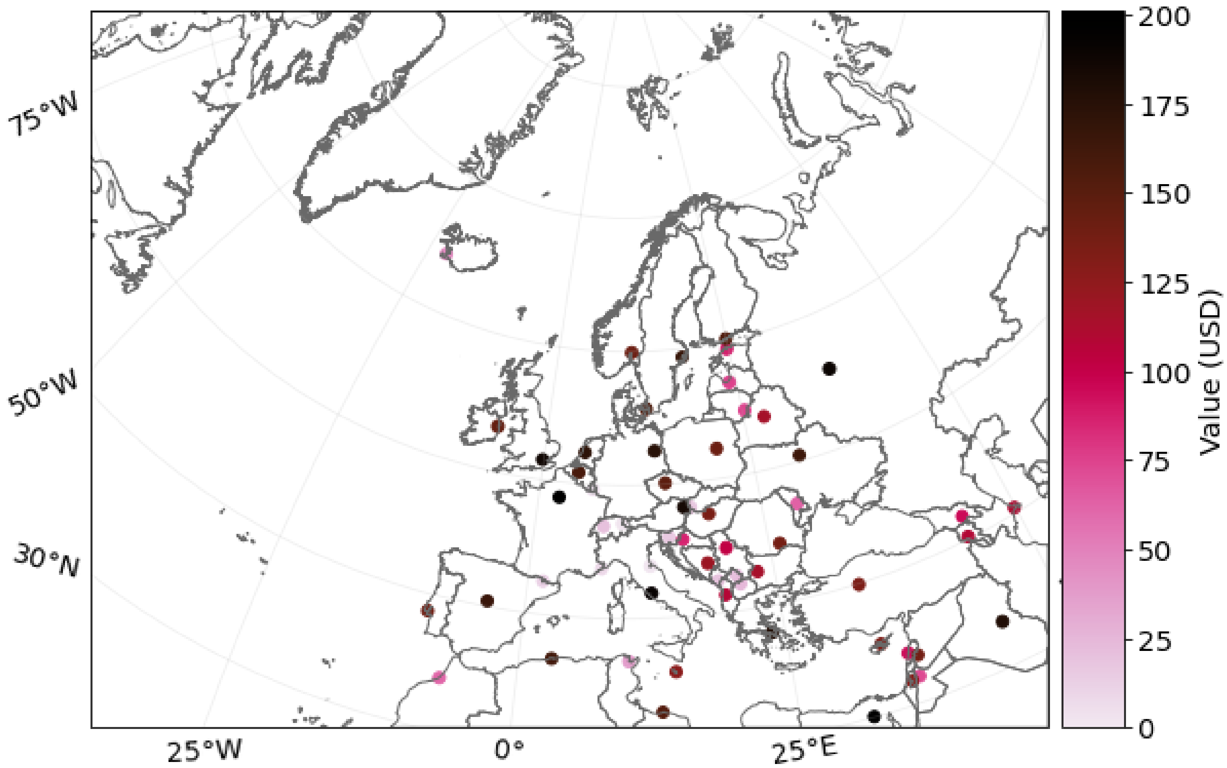
```
# Example 1: plot_hexbin method
print('\x1b[1;03;30;30m' + 'Plotting exp_df.' + '\x1b[0m')
axs = exp.plot_hexbin();

# further methods to check out:
# axs.set_xlim(15, 45) to modify x-axis borders, axs.set_ylim(10, 40) to modify y-
↪axis borders
# further keyword arguments to play around with: pop_name, buffer, gridsize, ...
```

```
Plotting exp_df.
```



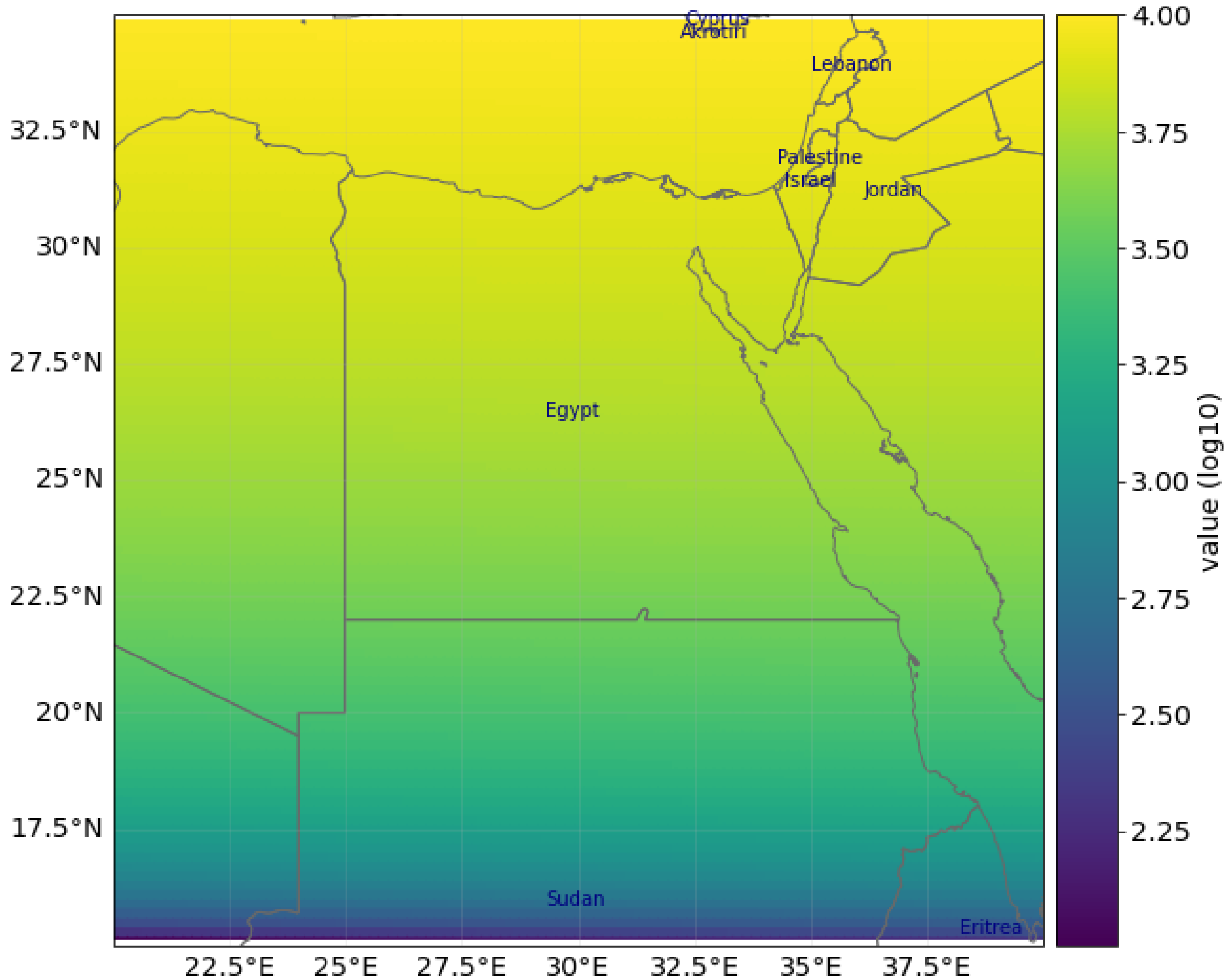

```python
# Example 2: plot_scatter method

exp_gpd.to_crs('epsg:3035', inplace=True)
exp_gpd.plot_scatter(pop_name=False);
```

```
<GeoAxesSubplot:>
```

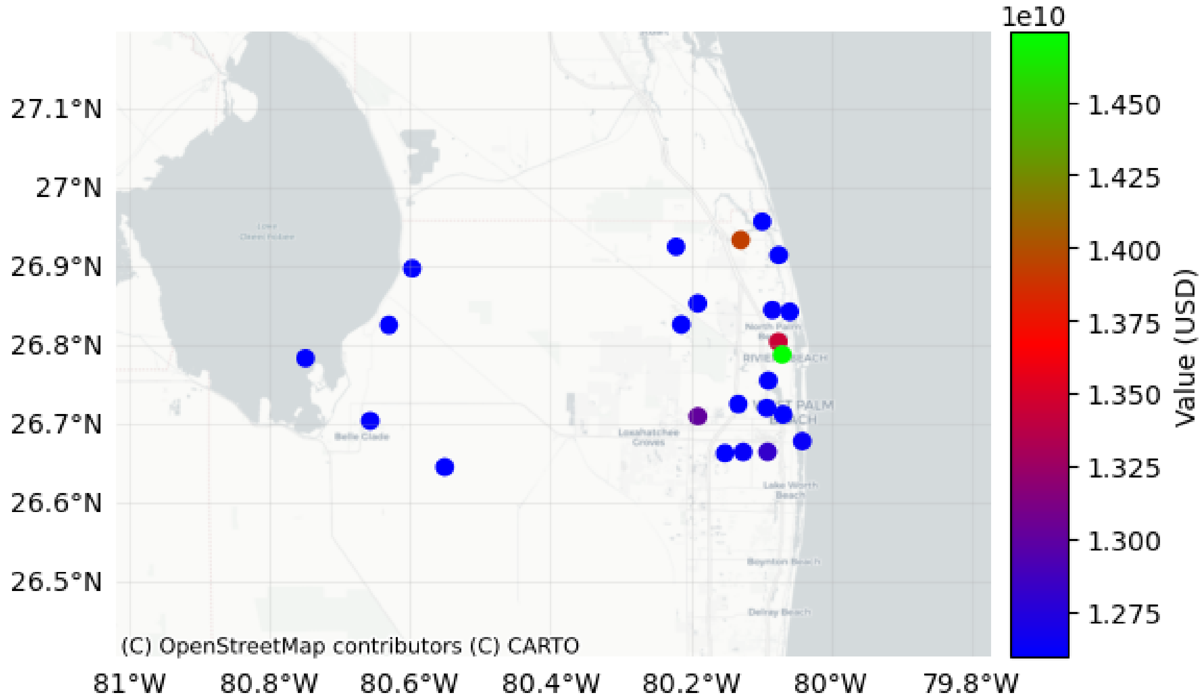


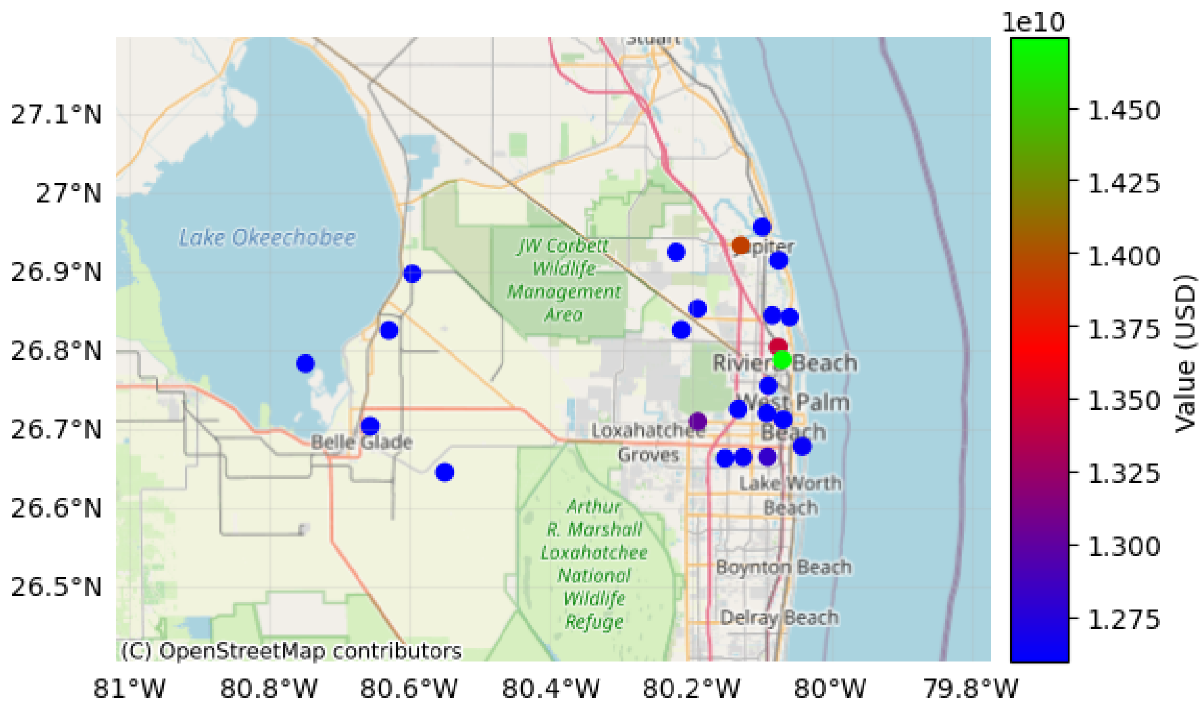

```
# Example 3: plot_raster method
from climada.util.plot import add_cntry_names # use climada's plotting utilities
ax = exp.plot_raster(); # plot with same resolution as data
add_cntry_names(ax, [exp.gdf.longitude.min(), exp.gdf.longitude.max(), exp.gdf.
↪latitude.min(), exp.gdf.latitude.max()])

# use keyword argument save_tiff='filepath.tiff' to save the corresponding raster in␣
↪tiff format
# use keyword argument raster_res='desired number' to change resolution of the raster.
```

```
2021-06-04 17:07:42,654 - climada.util.coordinates - INFO - Raster from resolution 0.
↪20202020202019355 to 0.20202020202019355.
```




```
# Example 4: plot_basemap method
import contextily as ctx
# select the background image from the available ctx.providers
ax = exp_templ.plot_basemap(buffer=30000, cmap='brg'); # using Positron from CartoDB
ax = exp_templ.plot_basemap(buffer=30000, cmap='brg',
                            url=ctx.providers.OpenStreetMap.Mapnik,  # Using
↪OpenStreetmap,
                            zoom=9);  # select the zoom level of the map, affects the
↪font size of labelled objects
```

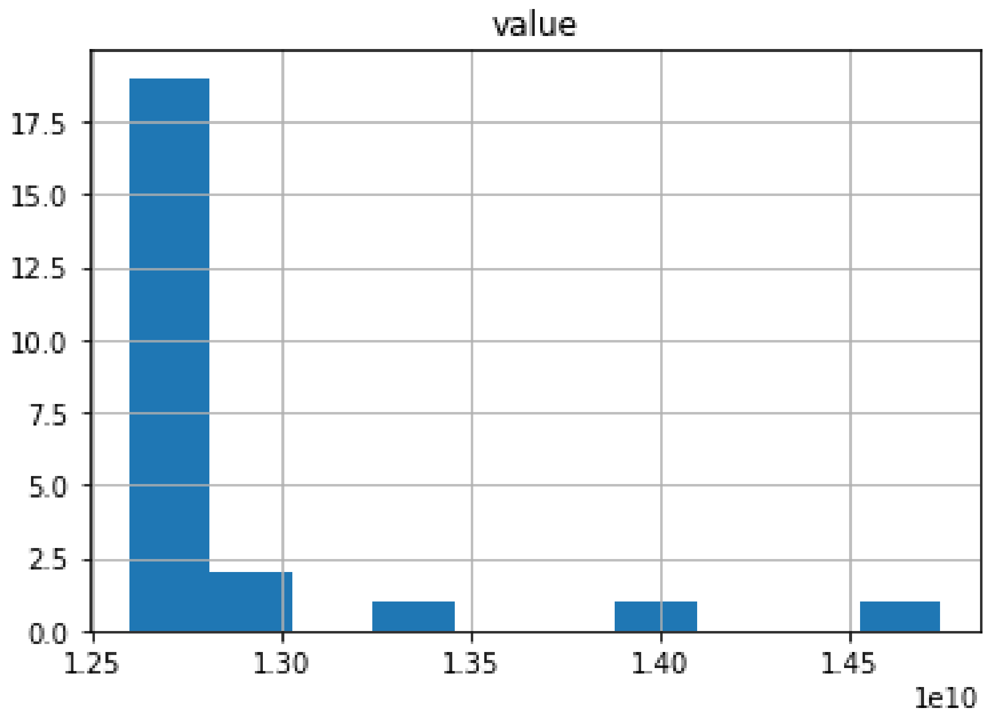




Since `Exposures` is a `GeoDataFrame`, any function for visualization from geopandas can be used. Check making maps and examples gallery.

```
# other visualization types
exp_templ.gdf.hist(column='value');
```

```
array([[<AxesSubplot:title={'center':'value'}>]], dtype=object)
```



9.1.5 Write (Save) Exposures

Exposures can be saved in any format available for `GeoDataFrame` (see fiona.supported_drivers) and `DataFrame` (pandas IO tools). Take into account that in many of these formats the metadata (e.g. variables `ref_year` and `value_unit`) will not be saved. Use instead the format hdf5 provided by `Exposures` methods `write_hdf5()` and `from_hdf5()` to handle all the data.

```
import fiona; fiona.supported_drivers
from climada import CONFIG
results = CONFIG.local_data.save_dir.dir()

# DataFrame save to csv format. geometry writen as string, metadata not saved!
exp_templ.gdf.to_csv(results.joinpath('exp_templ.csv'), sep='\t')
```

```
# write as hdf5 file
exp_templ.write_hdf5(results.joinpath('exp_temp.h5'))
```

Finallly, as with any Python object, use climada's save option to save it in pickle format.

```
# save in pickle format
from climada.util.save import save
# this generates a results folder in the current path and stores the output there
save('exp_templ.pkl.p', exp_templ) # creates results folder and stores there
```

9.1.6 Dask - improving performance for big exposure

Dask is used in some methods of CLIMADA and can be activated easily by proving the scheduler.

```
# set_geometry_points is expensive for big exposures
# for small amount of data, the execution time might be even greater when using dask
exp.gdf.drop(columns=['geometry'], inplace=True)
print(exp.gdf.head())
%time exp.set_geometry_points(scheduler='processes')
print(exp.gdf.head())
```

```
   value  latitude  longitude  impf_TC
0      0      15.0  20.000000        1
1      1      15.0  20.202020        1
2      2      15.0  20.404040        1
3      3      15.0  20.606061        1
4      4      15.0  20.808081        1
CPU times: user 243 ms, sys: 116 ms, total: 359 ms
Wall time: 2.52 s
   value  latitude  longitude  impf_TC                   geometry
0      0      15.0  20.000000        1  POINT (20.00000 15.00000)
1      1      15.0  20.202020        1  POINT (20.20202 15.00000)
2      2      15.0  20.404040        1  POINT (20.40404 15.00000)
3      3      15.0  20.606061        1  POINT (20.60606 15.00000)
4      4      15.0  20.808081        1  POINT (20.80808 15.00000)
```

9.2 LitPop class

9.2.1 Introduction

`LitPop` is an *Exposures*-type class. It is used to initiate gridded exposure data with estimates of either asset value, economic activity or population based on nightlight intensity and population count data.

Background

The modeling of economic disaster risk on a global scale requires high-resolution maps of exposed asset values. We have developed a generic and scalable method to downscale national asset value estimates proportional to a combination of nightlight intensity ("Lit") and population data ("Pop").

Asset exposure value is disaggregated to the grid points proportionally to $Lit^m Pop^n$, computed at each grid cell:

$Lit^m Pop^n = Lit^m * Pop^n$, with $exponents = [m, n] \in \mathbb{R}^+$ (Default values are $m = n = 1$).

For more information please refer to the related publication (https://doi.org/10.5194/essd-12-817-2020) and data archive (https://doi.org/10.3929/ethz-b-000331316).

How to cite: Eberenz, S., Stocker, D., Röösli, T., and Bresch, D. N.: *Asset exposure data for global physical risk assessment*, Earth Syst. Sci. Data, 12, 817–833, https://doi.org/10.5194/essd-12-817-2020, 2020.

Input data

Note: All required data except for the population data from Gridded Population of the World (GPW) is downloaded automatically when an `LitPop.set_*` method is called.

Warning: Processing the data for the first time can take up huge amounts of RAM (>10 GB), depending on country or region size. Consider using the *wrapper function* of the *data API* to download readily computed LitPop exposure data for default values ($n = m = 1$) on demand.

Nightlight intensity

Black Marble annual composite of the VIIRS day-night band (Grayscale) at 15 arcsec resolution is downloaded from the NASA Earth Observatory: https://earthobservatory.nasa.gov/Features/NightLights (available for 2012 and 2016 at 15 arcsec resolution (~500m)). The first time a nightlight image is used, it is downloaded and stored locally. This might take some time.

Population count

Gridded Population of the World (GPW), v4: Population Count, v4.10, v4.11 or later versions (2000, 2005, 2010, 2015, 2020), available from http://sedac.ciesin.columbia.edu/data/collection/gpw-v4/sets/browse.

The GPW file of the year closest to the requested year (`reference_year`) is required. To download GPW data a (free) login for the NASA SEDAC website is required.

Direct download links are avilable, also for older versions, i.e.:

- v4.11: http://sedac.ciesin.columbia.edu/downloads/data/gpw-v4/gpw-v4-population-count-rev11/gpw-v4-population-count-rev11_2015_30_sec_tif.zip
- v4.10: http://sedac.ciesin.columbia.edu/downloads/data/gpw-v4/gpw-v4-population-count-rev10/gpw-v4-population-count-rev10_2015_30_sec_tif.zip,
- Overview over all versions of GPW v.4: https://beta.sedac.ciesin.columbia.edu/data/collection/gpw-v4/sets/browse

The population data from GWP needs to be downloaded manually as TIFF from this site and placed in the `SYSTEM_DIR` folder of your climada installation.

Downloading existing LitPop asset exposure data

The easiest way to download existing data is using the *wrapper function* of the *data API*.

Readily computed LitPop asset exposure data based on Lit^1Pop^1 for 224 countries, distributing produced capital / non-financial wealth of 2014 at a resolution of 30 arcsec can also be downloaded from the ETH Research Repository: https://doi.org/10.3929/ethz-b-000331316. The dataset contains gridded data for more than 200 countries as CSV files.

9.2.2 Attributes

The `LitPop` class inherits from *Exposures*. It adds the following attributes:

```
    exponents : Defining powers (m, n) with which nightlights and population go into␣
↪Lit**m * Pop**n.
    fin_mode : Socio-economic indicator to be used as total asset value for␣
↪disaggregation.
    gpw_version : Version number of GPW population data, e.g. 11 for v4.11
```

fin_mode

The choice of `fin_mode` is crucial. Implemented choices are:

- `'pc'`: produced capital (Source: World Bank), incl. manufactured or built assets such as machinery, equipment, and physical structures. The pc-data is stored in the subfolder *data/system/Wealth-Accounts_CSV/*. Source: https://datacatalog.worldbank.org/dataset/wealth-accounting
- `'pop'`: population count (source: GPW, same as gridded population)
- `'gdp'`: gross-domestic product (Source: World Bank)
- `'income_group'`: gdp multiplied by country's income group+1
- `'nfw'`: non-financial household wealth (Source: Credit Suisse)
- `'tw'`: total household wealth (Source: Credit Suisse)
- `'norm'`: normalized, total value of country or region is 1.
- `'none'`: None – LitPop per pixel is returned unchanged

Regarding the GDP (nominal GDP at current USD) and income group values, they are obtained from the World Bank using the pandas-datareader API. If a value is missing, the value of the closest year is considered. When no values are provided from the World Bank, we use the Natural Earth repository values.

9.2.3 Key Methods

- `from_countries`: set exposure for one or more countries, see section `from_countries` below.
- `from_nightlight_intensity`: wrapper around `from_countries` and `from_shape` to load nightlight data to exposure.
- `from_population`: wrapper around `from_countries` and `from_shape_population` to load pure population data to exposure. This can be used to initiate a population exposure set.
- `from_shape_and_countries`: given a shape and a list of countries, exposure is initiated for the countries and then cropped to the shape. See section *Set custom shapes* below.
- `from_shape`: given any shape or geometry and an estimate of total values, exposure is initiated for the shape directly. See section *Set custom shapes* below.

```
# Import class LitPop:
from climada.entity import LitPop
```

9.2.4 `from_countries`

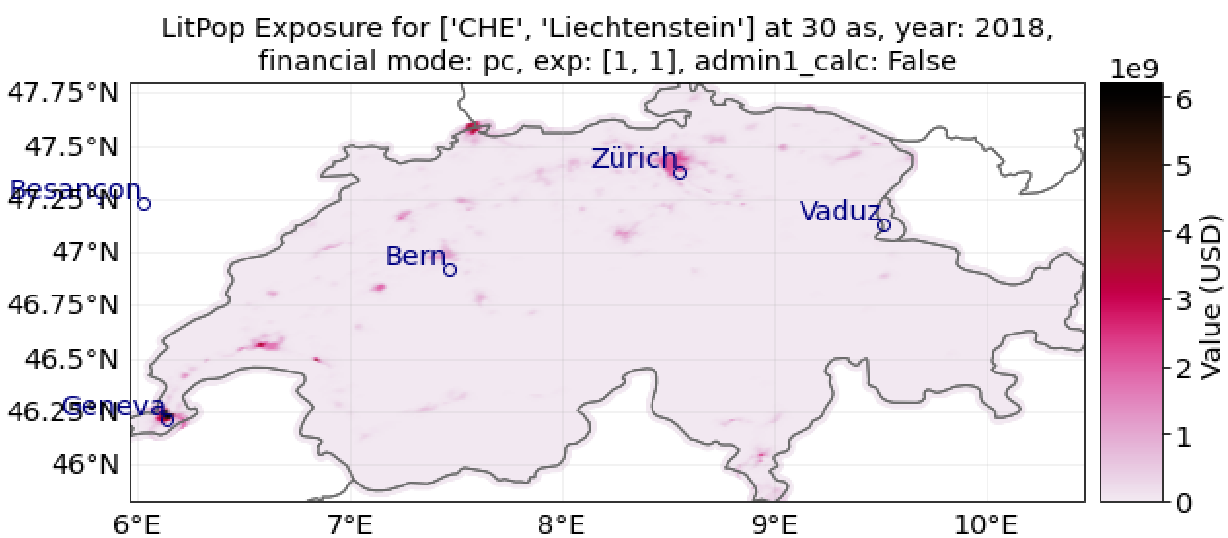
In the following, we will create exposure data sets and plots for a variety of countries, comparing different settings.

Default Settings

Per default, the exposure entity was initiated using the default parameters, i.e. a resolution of 30 arcsec, produced capital 'pc' as total asset value and using the exponents $(1, 1)$.

```
# Initiate a default LitPop exposure entity for Switzerland and Liechtenstein (ISO3-
↪Codes 'CHE' and 'LIE'):
try:
    exp = LitPop.from_countries(['CHE', 'Liechtenstein']) # you can provide either␣
↪single countries or a list of countries
except FileExistsError as err:
    print("Reason for error: The GPW population data has not been downloaded, c.f.␣
↪section 'Input data' above.")
    raise err
exp.plot_scatter();

# Note that `exp.gdf.region_id` is a number identifying each country:
print('\n Region IDs (`region_id`) in this exposure:')
print(exp.gdf.region_id.unique())
```

```
2021-10-19 17:03:20,108 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2021-10-19 17:03:23,876 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2021-10-19 17:03:23,953 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2021-10-19 17:03:24,476 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.

 Region IDs (`region_id`) in this exposure:
[756 438]
```



fin_mode, resolution and exponents

Instead on produced capital, we can also downscale other available macroeconomic indicators as estimates of asset value. The indicator can be set via the parameter `fin_mode`, either to 'pc', 'pop', 'gdp', 'income_group', 'nfw', 'tw', 'norm', or 'none'. See descriptions of each alternative above in the introduction.

We can also change the resolution via `res_arcsec` and the `exponents`.

The default resolution is 30 arcsec ≈ 1 km. A resolution of 3600 arcsec = 1 degree corresponds to roughly 110 km close to the equator.

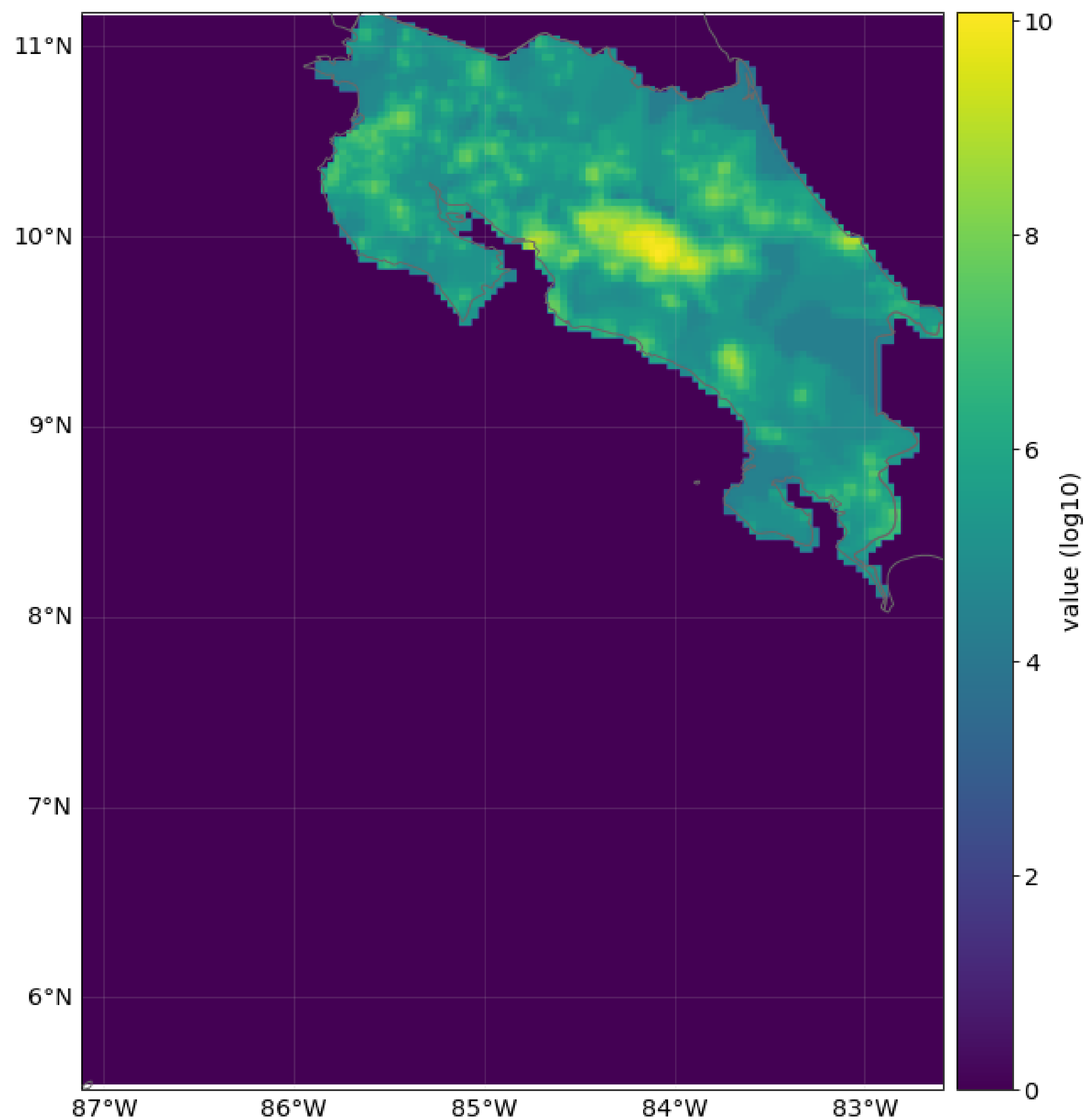
9.2.5 `from_population`

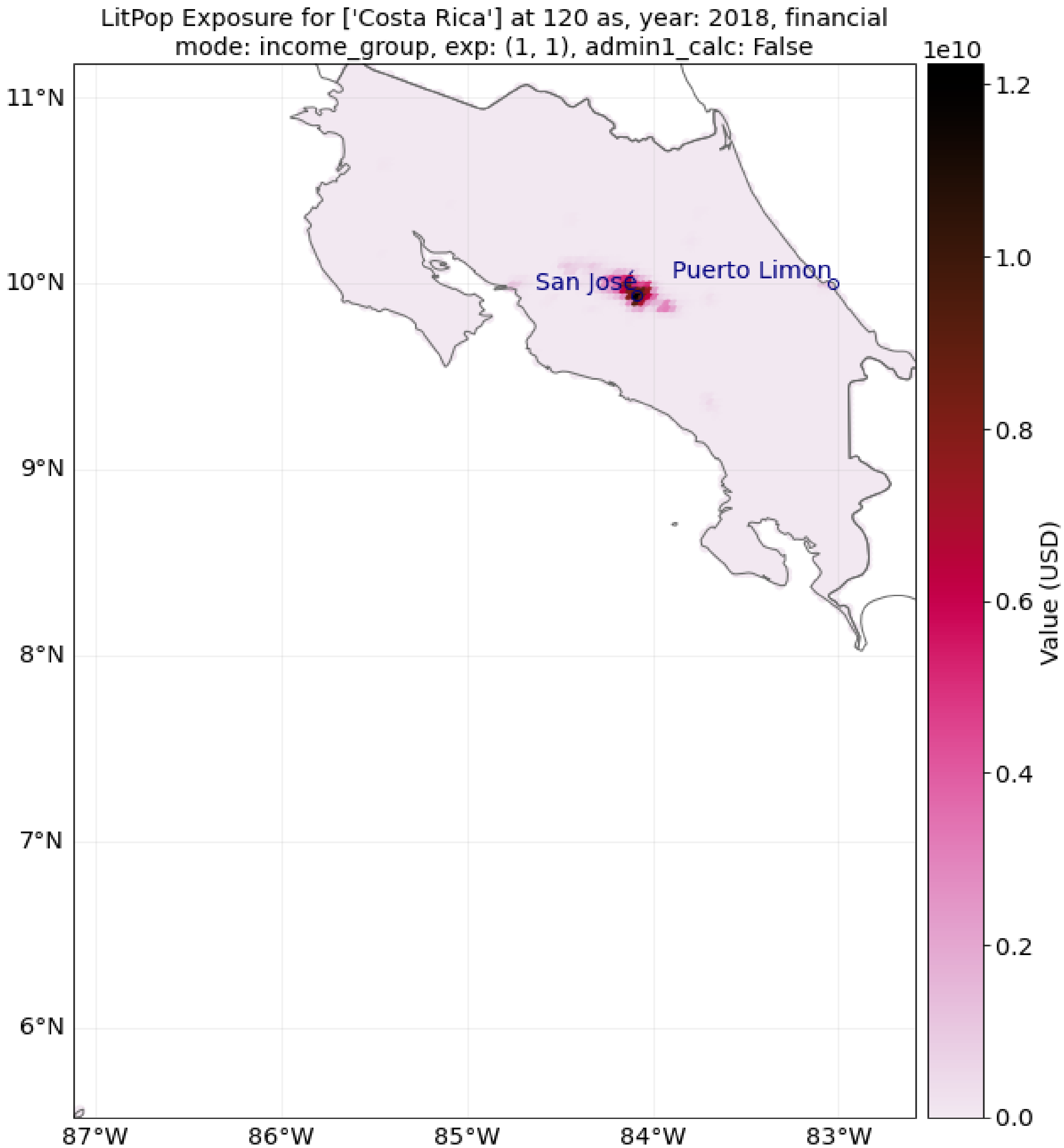
Let's initiate an exposure instance with the financial mode "income_group" and at a resolution of 120 arcsec (roughly 4 km).

```
# Initiate a LitPop exposure entity for Costa Rica with varied resolution, fin_mode,␣
↪and exponents:
exp = LitPop.from_countries('Costa Rica', fin_mode='income_group', res_arcsec=120,␣
↪exponents=(1,1)) # change the parameters and see what happens...
# exp = LitPop.from_countries('Costa Rica', fin_mode='gdp', res_arcsec=90,␣
↪exponents=(3,0)) # example of variation
exp.plot_raster(); # note the log scale of the colorbar
exp.plot_scatter();
```

```
2021-10-19 17:03:26,671 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
```

```
<GeoAxesSubplot:title={'center':"LitPop Exposure for ['Costa Rica'] at 120 as, year:␣
↪2018, financial\nmode: income_group, exp: (1, 1), admin1_calc: False"}>
```


11°N
10°N
9°N
8°N
7°N
6°N
87°W
86°W
85°W
84°W
83°W
10
8
6
4
2
0
value (log10)



9.2.6 Reference year

Additionally, we can change the year our exposure is supposed to represent. For this, nightlight and population data are used that are closest to the requested years. Macroeconomic indicators like produced capital are interpolated from available data or scaled proportional to GDP.

Let's load a population exposure map for Switzerland in 2000 and 2021 with a resolution of 300 arcsec:

```
# You may want to check if you have downloaded dataset Gridded Population of the␣
↪World (GPW), v4: Population Count, v4.11
```

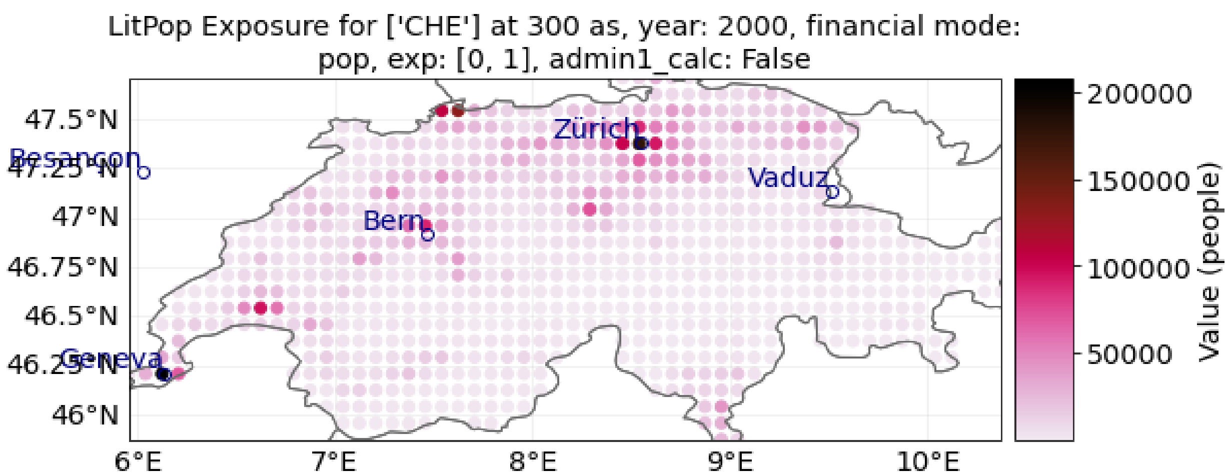
(continues on next page)

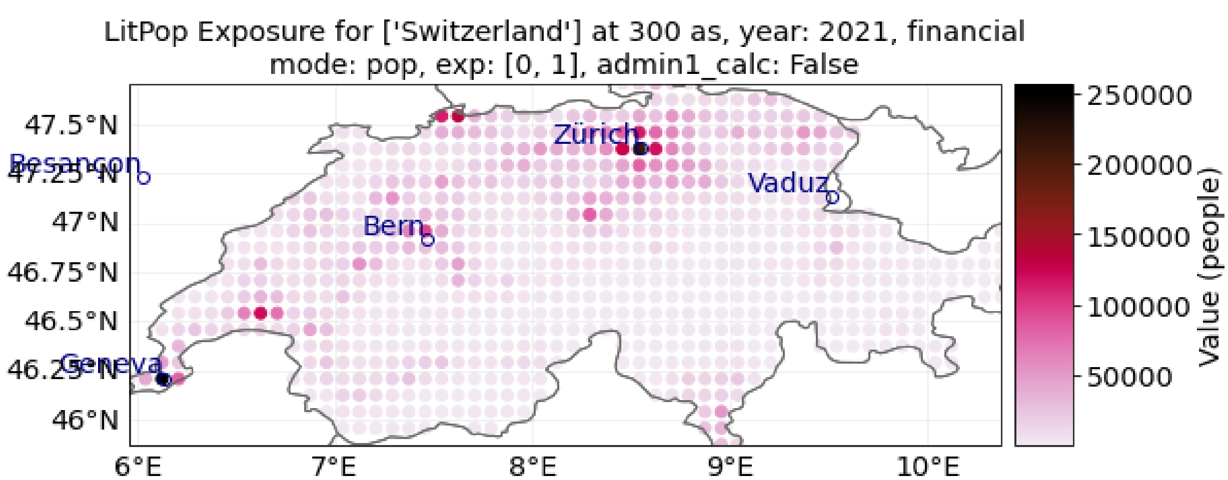
(continued from previous page)

```
# (2000 and 2020) first
pop_2000 = LitPop.from_countries('CHE', fin_mode='pop', res_arcsec=300, exponents=(0,
↪1), reference_year=2000)
# Alternatively, we ca use `from_population`:
pop_2021 = LitPop.from_population(countries='Switzerland', res_arcsec=300, reference_
↪year=2021)
# Since no population data for 2021 is available, the closest data point, 2020, is␣
↪used (see LOGGER.warning)
pop_2000.plot_scatter();
pop_2021.plot_scatter();
"""Note the difference in total values on the color bar."""
```

```
2021-10-19 17:03:31,884 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2021. Using nearest available year for GPW data: 2020
```

```
'Note the difference in total values on the color bar.'
```





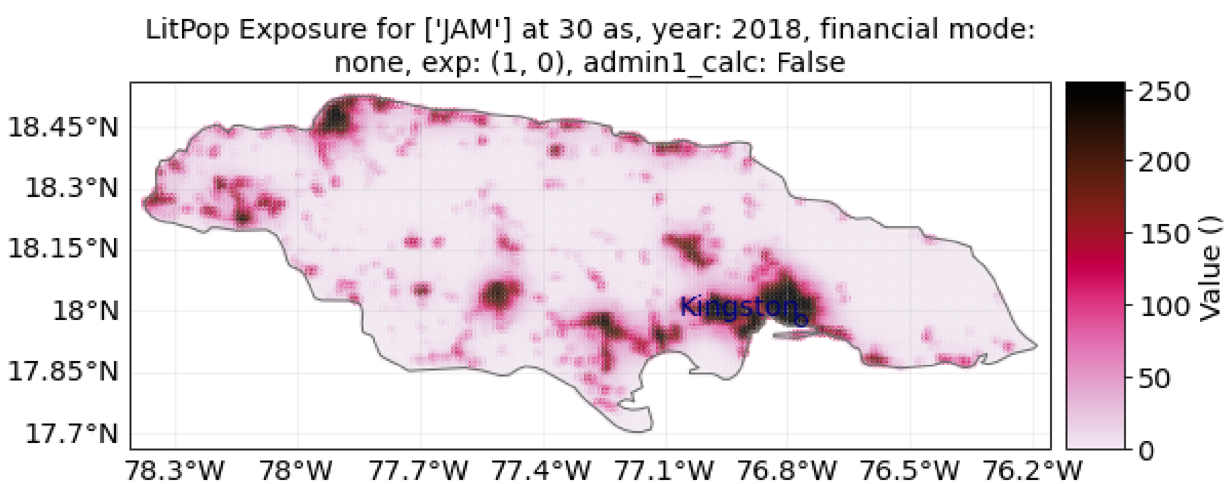
9.2.7 `from_nightlight_intensity` and `from_population`

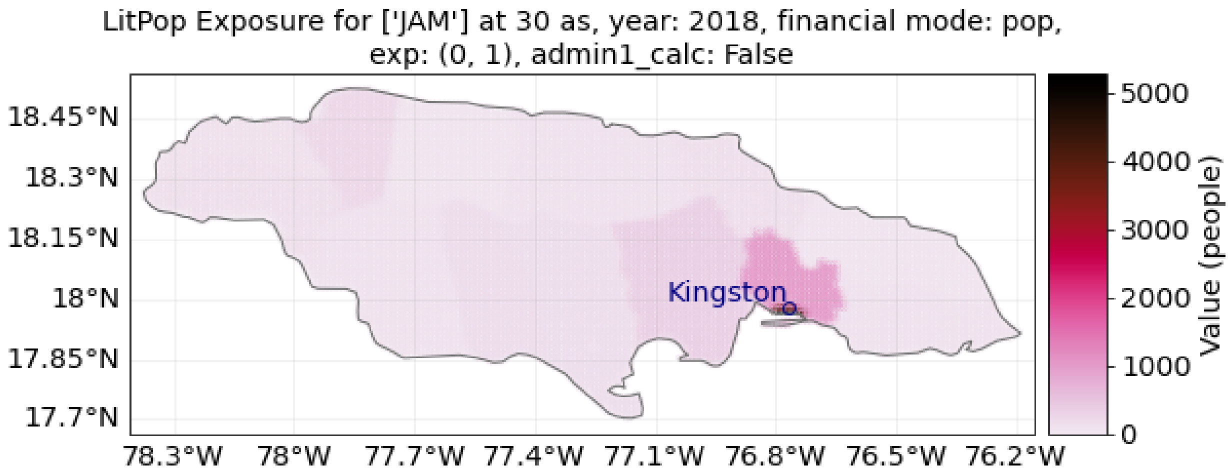
These wrapper methods can be used to produce exposures that are showing purely nightlight intensity or purely population count.

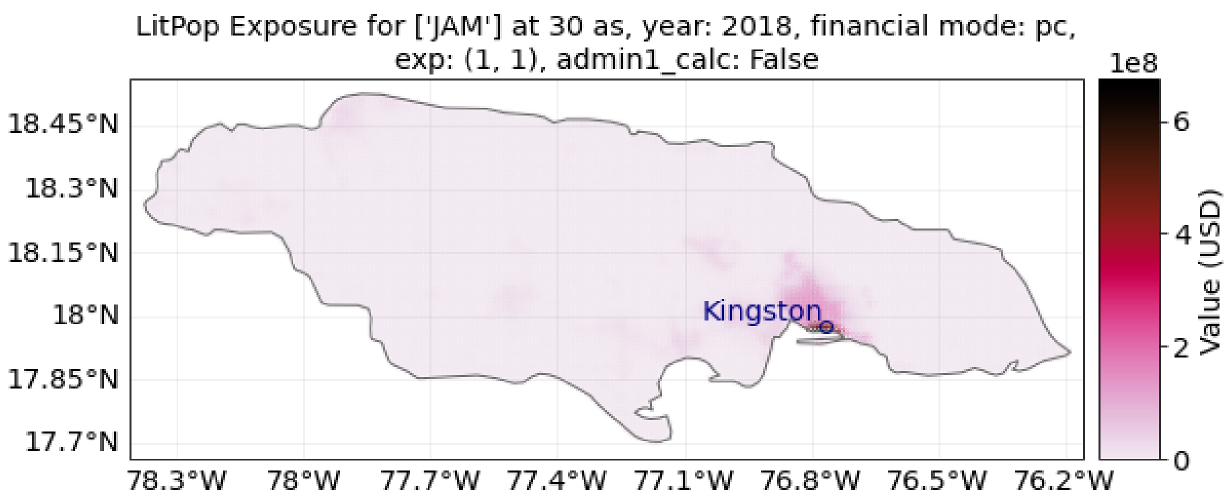
```
res = 30 # If you don't get an output after a very long time with country = "MEX",␣
↪try with res = 100
country = 'JAM' # Try different countries, i.e. 'JAM', 'CHE', 'RWA', 'MEX'
markersize = 4 # for plotting
buffer_deg=.04

exp_nightlights = LitPop.from_nightlight_intensity(countries=country, res_arcsec=res)
↪# nightlight intensity
exp_nightlights.plot_hexbin(linewidth=markersize, buffer=buffer_deg);
# Compare to the population map:
exp_population = LitPop().from_population(countries=country, res_arcsec=res)
exp_population.plot_hexbin(linewidth=markersize, buffer=buffer_deg);
# Compare to default LitPop exposures:
exp = LitPop.from_countries(countries=country, res_arcsec=res)
exp.plot_hexbin(linewidth=markersize, buffer=buffer_deg);
```

```
2021-10-19 17:03:34,705 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2021-10-19 17:03:35,308 - climada.entity.exposures.litpop.litpop - WARNING - Note:␣
↪set_nightlight_intensity sets values to raw nightlight intensity, not to USD. To␣
↪disaggregate asset value proportionally to nightlights^m, call from_countries or␣
↪from_shape with exponents=(m,0).
2021-10-19 17:03:39,867 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2021-10-19 17:03:44,032 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
```

```
<GeoAxesSubplot:title={'center':"LitPop Exposure for ['JAM'] at 30 as, year: 2018,␣
↪financial mode: pc,\nexp: (1, 1), admin1_calc: False"}>
```







For **Switzerland**, population is resolved on the 3rd administrative level, with 2538 distinct geographical units. Therefore, the purely population-based map is highly resolved.

For **Jamaica**, population is only resolved on the 1st administrative level, with only 14 distinct geographical units. Therefore, the purely population-based map shows large monotonous patches. The combination of Lit and Pop results in a concentration of asset value estimates around the capital city Kingston.

9.2.8 Init LitPop-Exposure from custom shapes

The methods `LitPop.from_shape_and_countries` and `LitPop.from_shape` initiate a LitPop-exposure instance for a given custom shape instead of a country. This can be used to initiate exposure for admin1-regions, i.e. cantons, states, districts, - but also for bounding boxes etc.

The difference between the two methods is that for `from_shape_and_countries`, the exposure for one or more whole countries is initiated first and then it is cropped to the shape. Please make sure that the shape is contained in the given countries. With `from_shape`, the shape is initiated directly which is much more resource efficient but requires a `total_value` to be provided by the user.

A population exposure for a custom shape can be initiated directly via `from_population` without providing `total_value`.

Example: State of Florida

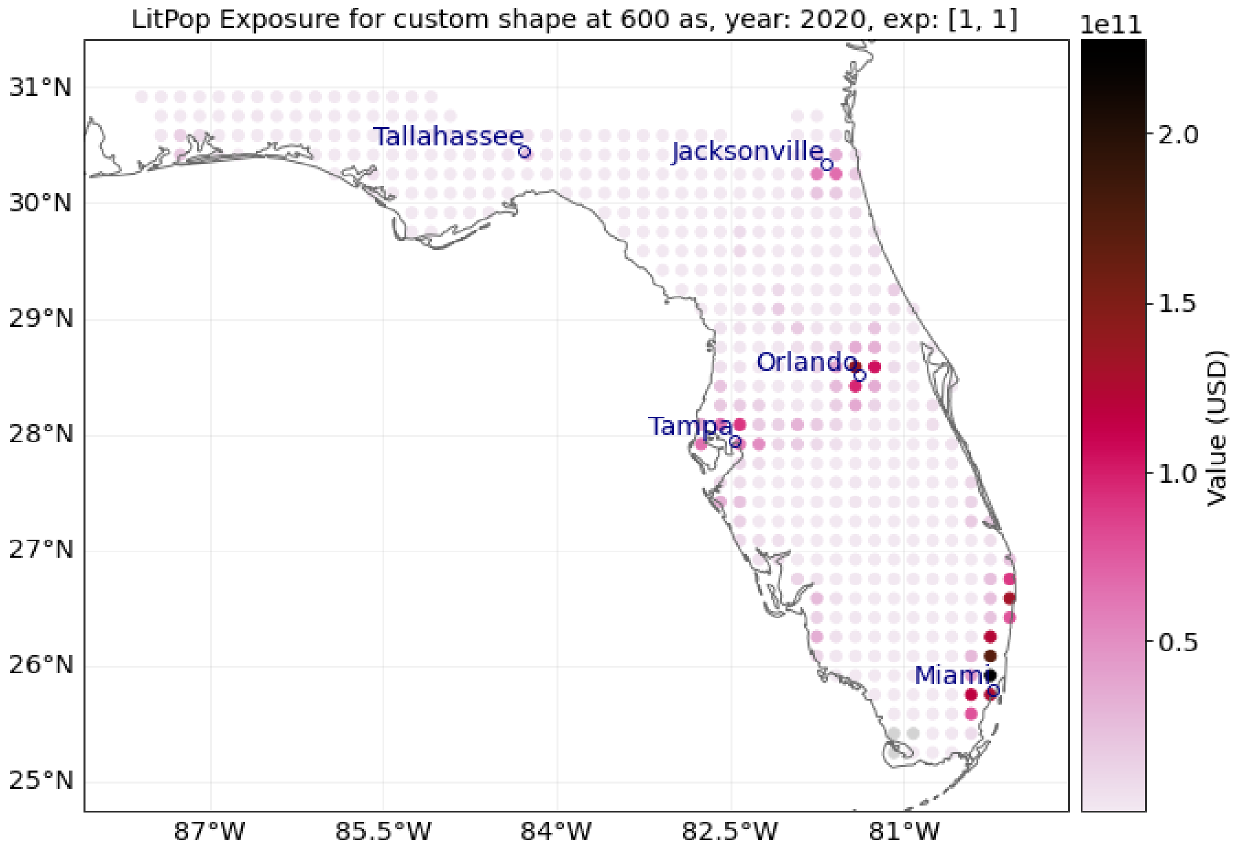
Using `LitPop.from_shape_and_countries` and `LitPop.from_shape` we initiate LitPop exposures for Florida:

```
import time
import climada.util.coordinates as u_coord
import climada.entity.exposures.litpop as lp

country_iso3a = 'USA'
state_name = 'Florida'
reslution_arcsec = 600
"""First, we need to get the shape of Florida:"""
admin1_info, admin1_shapes = u_coord.get_admin1_info(country_iso3a)
admin1_info = admin1_info[country_iso3a]
admin1_shapes = admin1_shapes[country_iso3a]
admin1_names = [record['name'] for record in admin1_info]
print(admin1_names)
for idx, name in enumerate(admin1_names):
    if admin1_names[idx]==state_name:
        break
print('Florida index: ' + str(idx))

"""Secondly, we estimate the `total_value`"""
# `total_value` required user input for `from_shape`, here we assume 5% of total␣
↪value of the whole USA:
total_value = 0.05 * lp._get_total_value_per_country(country_iso3a, 'pc', 2020)

"""Then, we can initiate the exposures for Florida:"""
start = time.process_time()
exp = LitPop.from_shape(admin1_shapes[idx], total_value, res_arcsec=600, reference_
↪year=2020)
print(f'\n Runtime `from_shape` : {time.process_time() - start:1.2f} sec.\n')
exp.plot_scatter(vmin=100, buffer=.5);
```

```
['Minnesota', 'Washington', 'Idaho', 'Montana', 'North Dakota', 'Michigan', 'Maine',
↪'Ohio', 'New Hampshire', 'New York', 'Vermont', 'Pennsylvania', 'Arizona',
↪'California', 'New Mexico', 'Texas', 'Alaska', 'Louisiana', 'Mississippi', 'Alabama
↪', 'Florida', 'Georgia', 'South Carolina', 'North Carolina', 'Virginia', 'District␣
↪of Columbia', 'Maryland', 'Delaware', 'New Jersey', 'Connecticut', 'Rhode Island',
↪'Massachusetts', 'Oregon', 'Hawaii', 'Utah', 'Wyoming', 'Nevada', 'Colorado',
↪'South Dakota', 'Nebraska', 'Kansas', 'Oklahoma', 'Iowa', 'Missouri', 'Wisconsin',
↪'Illinois', 'Kentucky', 'Arkansas', 'Tennessee', 'West Virginia', 'Indiana']
Florida index: 20
```

```
 Runtime `from_shape` : 9.01 sec.
```

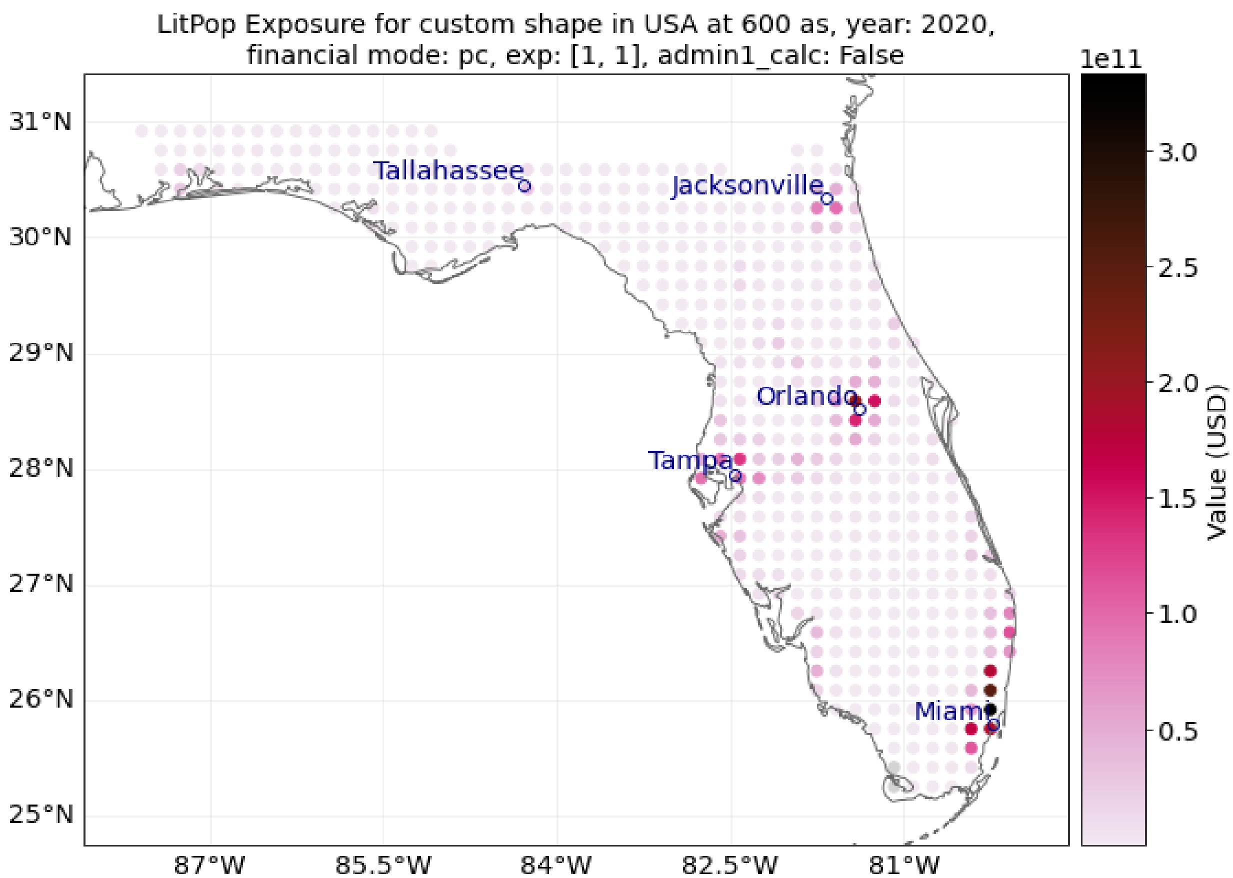
```
<GeoAxesSubplot:title={'center':'LitPop Exposure for custom shape at 600 as, year:␣
↪2020, exp: [1, 1]'}>
```




```
# `from_shape_and_countries` does not require `total_value`, but is slower to compute␣
↪than `from_shape`,
# because first, the exposure for the whole USA is initiated:
start = time.process_time()
exp = LitPop.from_shape_and_countries(admin1_shapes[idx], country_iso3a, res_
↪arcsec=600, reference_year=2020)
print(f'\n Runtime `from_shape_and_countries` : {time.process_time() - start:1.2f}␣
↪sec.\n')
exp.plot_scatter(vmin=100, buffer=.5);
"""Note the differences in computational speed and total value between the two␣
↪approaches"""
```

```
Runtime `from_shape_and_countries` : 24.49 sec.
```

```
'Note the differences in computational speed and total value between the two␣
↪approaches'
```



9.2.9 Example: Zurich city area

You can also define your own shape as a Polygon:

```
import time
from shapely.geometry import Polygon

"""initiate LitPop exposures for a geographical box around the city of Zurich:"""
bounds = (8.41, 47.25, 8.70, 47.47) # (min_lon, max_lon, min_lat, max_lat)
total_value=1000 # required user input for `from_shape`, here we just assume USD 1000␣
↪of total value
shape = Polygon([
                (bounds[0], bounds[3]),
                (bounds[2], bounds[3]),
                (bounds[2], bounds[1]),
                (bounds[0], bounds[1])
                ])
import time
start = time.process_time()
exp = LitPop.from_shape(shape, total_value)
print(f'\n Runtime `from_shape` : {time.process_time() - start:1.2f} sec.\n')
exp.plot_scatter();
# `from_shape_and_countries` does not require `total_value`, but is slower to compute:
start = time.process_time()
exp = LitPop.from_shape_and_countries(shape, 'Switzerland')
```

(continues on next page)

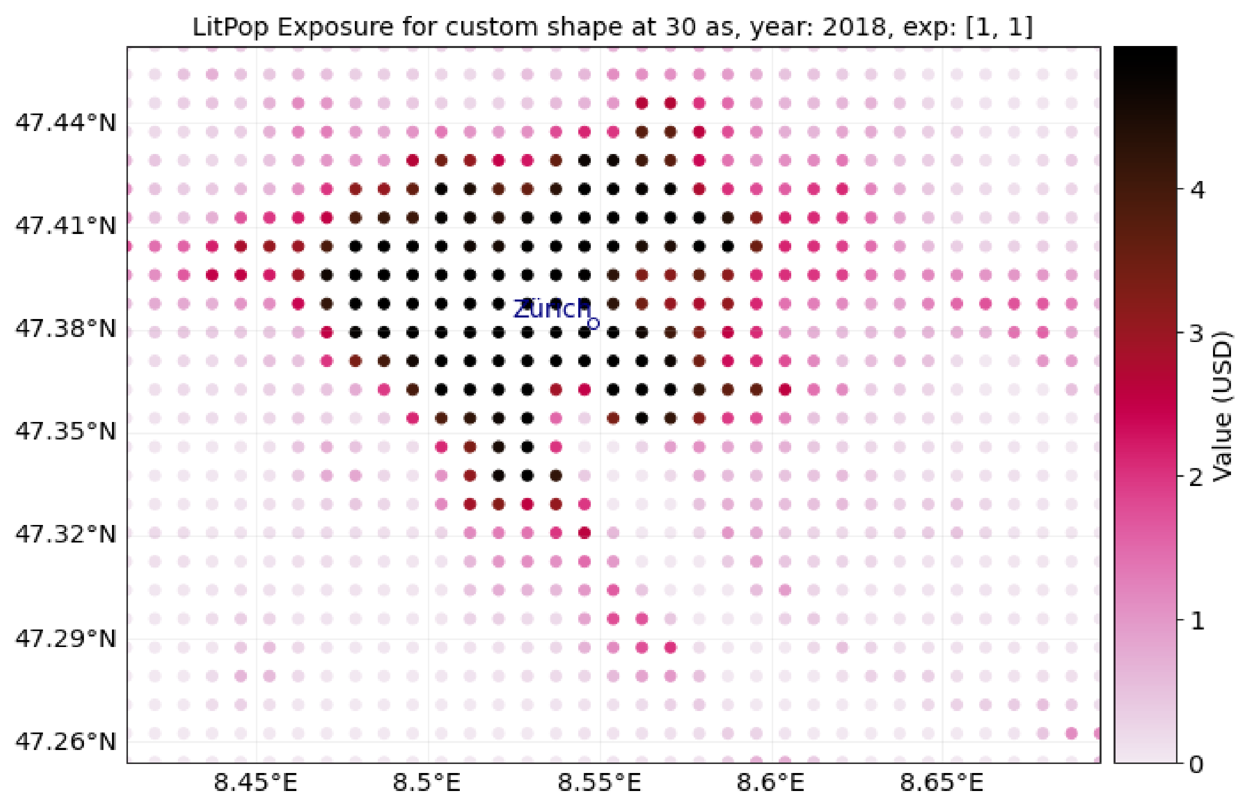
(continued from previous page)

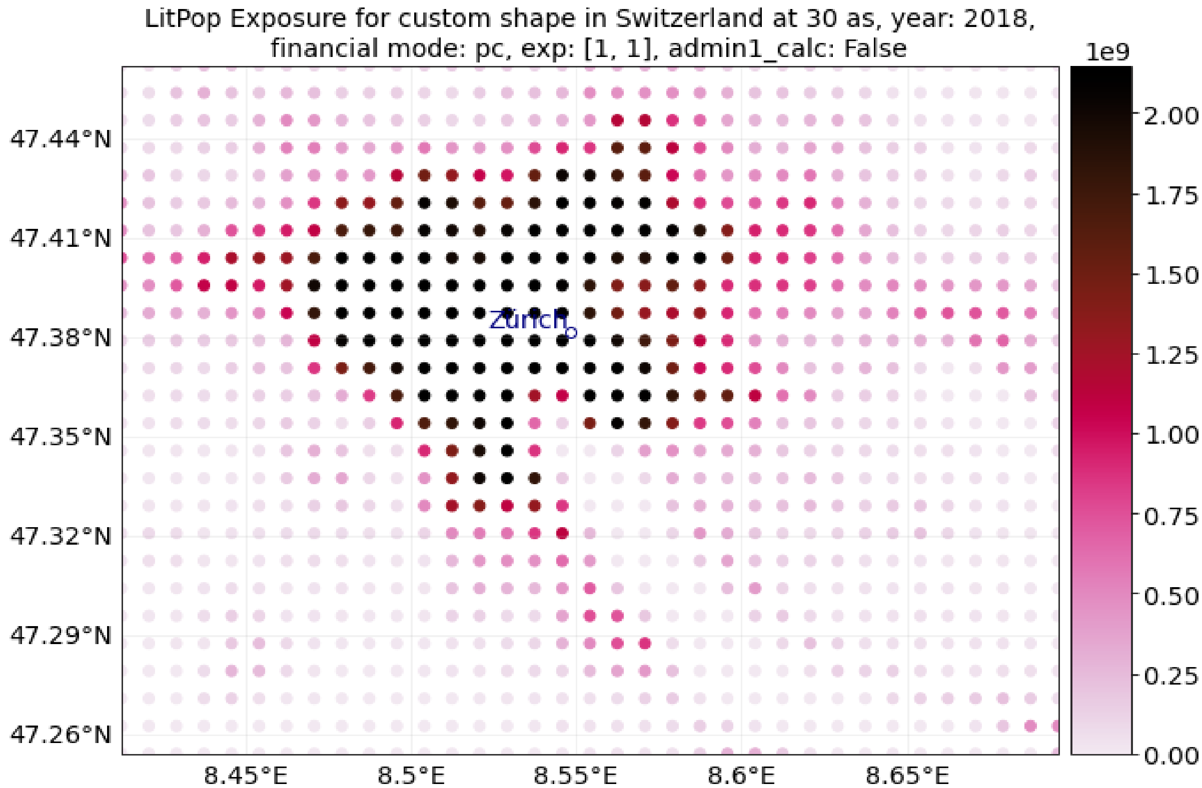
```
print(f'\n Runtime `from_shape_and_countries` : {time.process_time() - start:1.2f}␣
↪sec.\n')
exp.plot_scatter();
"""Note the difference in total value between the two exposure sets!"""

"""For comparison, initiate population exposure for a geographical box around the␣
↪city of Zurich:"""
start = time.process_time()
exp_pop = LitPop.from_population(shape=shape)
print(f'\n Runtime `from_population` : {time.process_time() - start:1.2f} sec.\n')
exp_pop.plot_scatter();

"""Population exposure for a custom shape can be initiated directly via `set_
↪population` without providing `total_value`"""
```

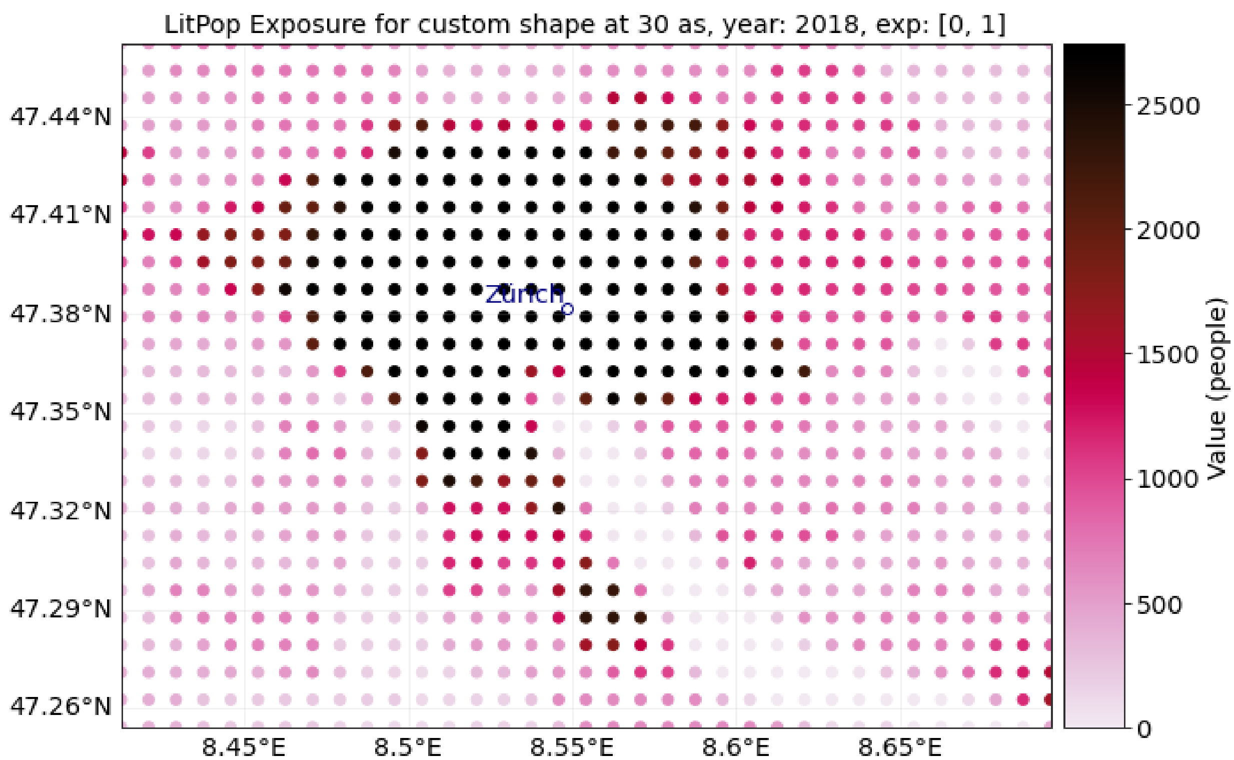
```
 Runtime `from_shape` : 0.51 sec.

2021-10-19 17:04:24,606 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020

 Runtime `from_shape_and_countries` : 3.18 sec.
```

```
 Runtime `from_population` : 0.75 sec.
```

```
'Population exposure for a custom shape can be initiated directly via `set_
↪population` without providing `total_value`'
```




LitPop Exposure for custom shape in Switzerland at 30 as, year: 2018,
financial mode: pc, exp: [1, 1], admin1_calc: False
1e9
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Value (USD)
47.44°N
47.41°N
47.38°N
47.35°N
47.32°N
47.29°N
47.26°N
8.45°E
8.5°E
8.55°E
8.6°E
8.65°E
Zürich


LitPop Exposure for custom shape at 30 as, year: 2018, exp: [0, 1]
2500
2000
1500
1000
500
0
Value (people)
47.44°N
47.41°N
47.38°N
47.35°N
47.32°N
47.29°N
47.26°N
8.45°E
8.5°E
8.55°E
8.6°E
8.65°E
Zürich

9.2.10 Sub-national (admin-1) GDP as intermediate downscaling layer

In order to improve downscaling for countries with large regional differences within, a subnational breakdown of GDP can be used as an intermediate downscaling layer wherever available.

The sub-national (admin-1) GDP-breakdown needs to be added manually as a ".xls"-file to the folder `data/system/GSDP` in the CLIMADA-directory. Currently, such data is provided for more than 10 countries, including USA, India, and China.

The xls-file requires at least the following columns (with names specified in row 1):

- `State_Province`: Names of admin-1 regions, i.e. states, cantons, provinces. Names need to match the naming of admin-1 shapes in the data used by the python package `cartopy.io` (c.f. `shapereader.natural_earth(name='admin_1_states_provinces')`)
- `GSDP_ref`: value of sub-national GDP to be used (absolute or relative values)
- `Postal`, optional: Alternative identifier of region, if names do not match with cartopy. Needs to correspond to the Postal-identifiers used in the shapereader of `cartopy.io`.

Please note that while admin1-GDP will per definition improve the downscaling of *GDP*, it might not necessarily improve the downscaling quality for other asset bases like produced capital (pc).

How To:

The intermediate downscaling layer can be activated with the parameter `admin1_calc=True`.

```
# Initiate GDP-Entity for Switzerland, with and without admin1_calc:

ent_adm0 = LitPop.from_countries('CHE', res_arcsec=120, fin_mode='gdp', admin1_
↪calc=False)
ent_adm0.set_geometry_points()

ent_adm1 = LitPop.from_countries('CHE', res_arcsec=120, fin_mode='gdp', admin1_
↪calc=True)

ent_adm0.check()
ent_adm1.check()
print('Done.')
```

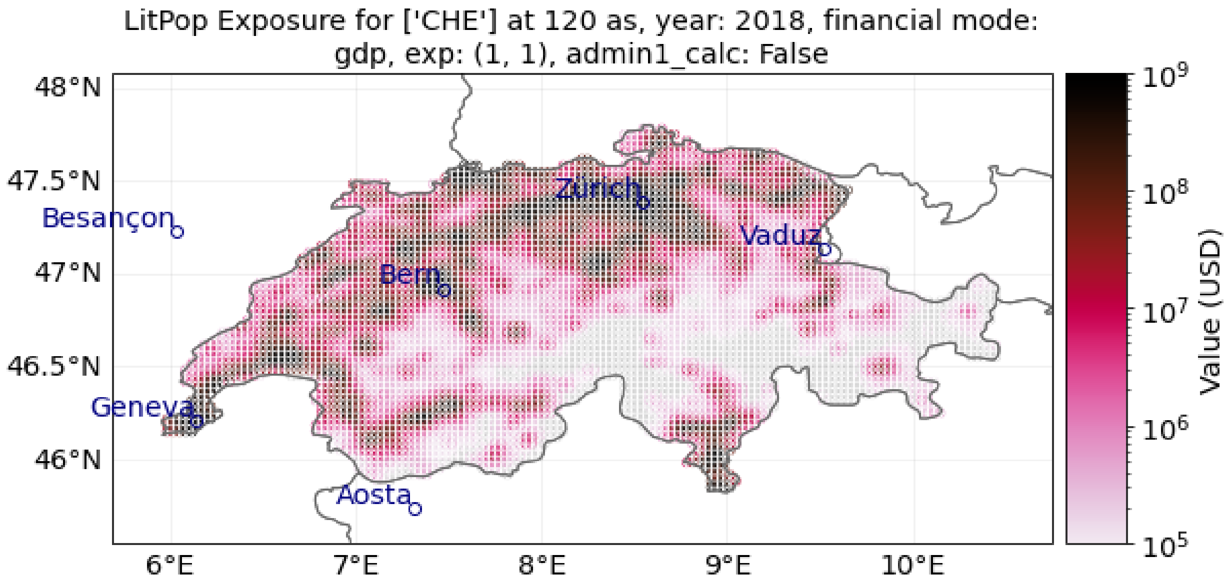
```
2021-10-19 17:04:31,363 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
```

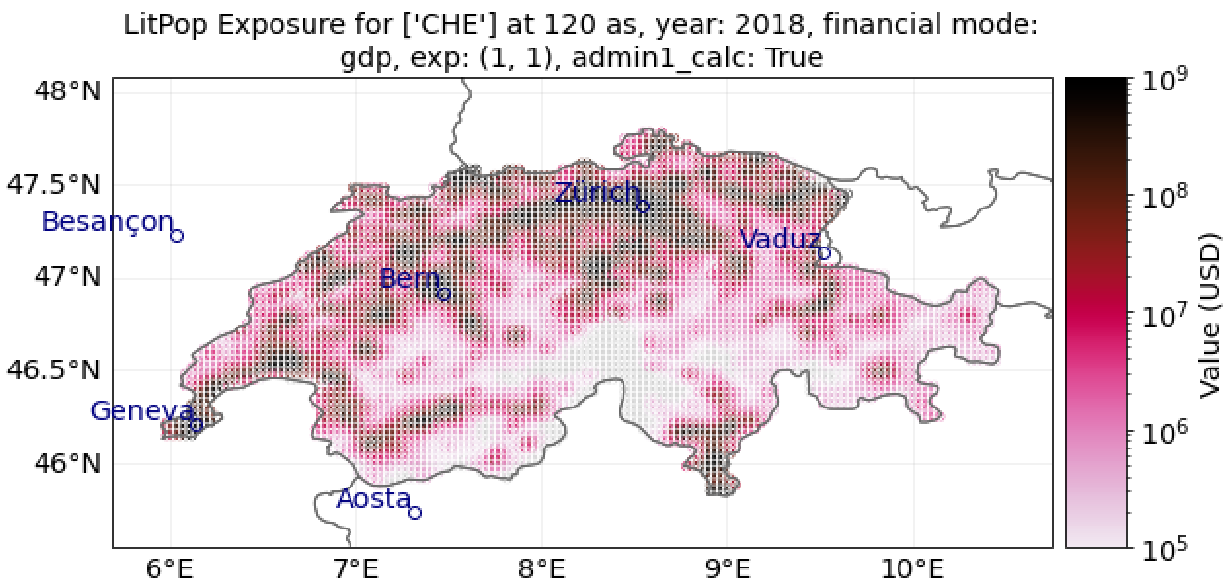
```
Done.
```

```
# Plotting:
from matplotlib import colors
norm=colors.LogNorm(vmin=1e5, vmax=1e9) # setting range for the log-normal scale
markersize = 5
ent_adm0.plot_hexbin(buffer=.3, norm=norm, linewidth=markersize);
ent_adm1.plot_hexbin(buffer=.3, norm=norm, linewidth=markersize);

print('admin-0: First figure')
print('admin-1: Second figure')
'''Do you spot the small differences in Graubünden (eastern Switzerland)?'''
```

```
admin-0: First figure
admin-1: Second figure
```

```
'Do you spot the small differences in Graubünden (eastern Switzerland)?'
```





9.3 How to use polygons or lines as exposure

9.3.1 Contents

- *Introduction*
- *Quick example*
- *Polygons*
- *Lines*

9.3.2 Introduction

Exposure in CLIMADA are usually represented as individual points or a raster of points. See *Exposures* tutorial to learn how to fill and use exposures. In this tutorial we show you how to run the impact calculation chain if you have polygons or lines to start with. The approach provides an all-in-one method for impact calculation: `calc_geom_impact` It features three sub-steps, for which the current util module `lines_polys_handler` also provides separate functions:

1. Disaggregation of line and polygon data into point exposure:
   - Interpolate geometries to points to fit in an `Exposure` instance;
   - Disaggregate the respective geometry values to the point values
2. Perform the impact calculation in CLIMADA with the point exposure
3. Aggregate the calculated point `Impact` back to an impact instance for the initial polygons or lines

9.3.3 Quick example

Get example polygons (provinces), lines (rails), points exposure for the Netherlands, and create one single Exposures. Get demo winter storm hazard and a corresponding impact function.

```
from climada.util.api_client import Client
import climada.util.lines_polys_handler as u_lp
from climada.entity.impact_funcs import ImpactFuncSet
from climada.entity.impact_funcs.storm_europe import ImpfStormEurope
from climada.entity import Exposures

HAZ = Client().get_hazard('storm_europe', name='test_haz_WS_nl', status='test_dataset
↪');

EXP_POLY = Client().get_exposures('base', name='test_polygon_exp', status='test_
↪dataset');
EXP_LINE = Client().get_exposures('base', name='test_line_exp', status='test_dataset
↪');
EXP_POINT = Client().get_exposures('base', name='test_point_exp', status='test_dataset
↪');

EXP_MIX = Exposures.concat([EXP_POLY, EXP_LINE, EXP_POINT])

IMPF = ImpfStormEurope.from_welker()
IMPF_SET = ImpactFuncSet([IMPF])
```

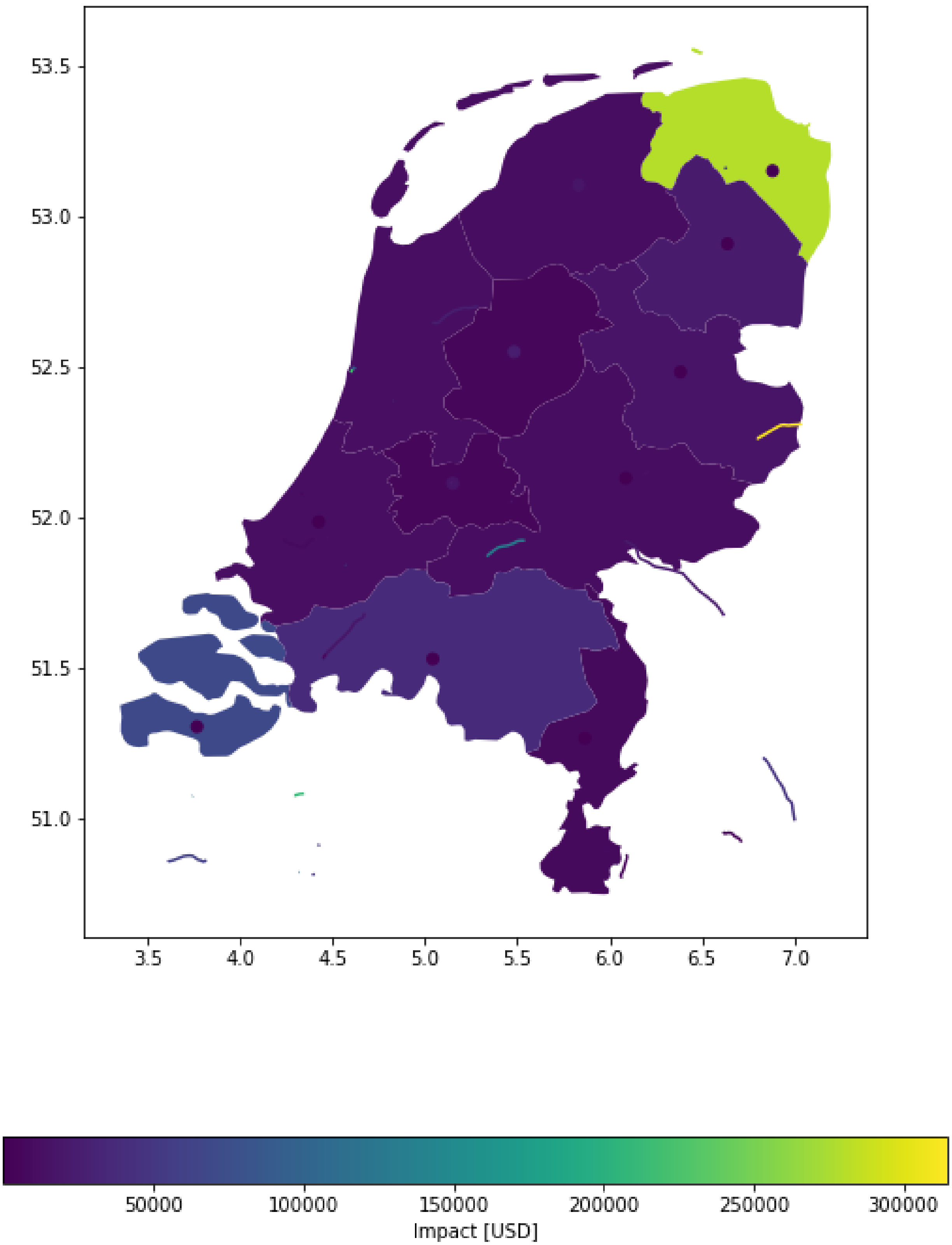
Compute the impact in one line.

```
#disaggregate in the same CRS as the exposures are defined (here degrees), resolution␣
↪1degree
#divide values on points
#aggregate by summing

impact = u_lp.calc_geom_impact(
            exp=EXP_MIX, impf_set=IMPF_SET, haz=HAZ,
            res=0.2, to_meters=False, disagg_met=u_lp.DisaggMethod.DIV, disagg_
↪val=None,
            agg_met=u_lp.AggMethod.SUM
            )
```

```
2022-06-24 13:16:15,277 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:16:15,284 - climada.entity.exposures.base - INFO - Matching 183␣
↪exposures with 9944 centroids.
2022-06-24 13:16:15,285 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:15,295 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:16:15,296 - climada.engine.impact - INFO - Calculating damage for 182␣
↪assets (>0) and 2 events.
```

```
/Users/ckropf/Documents/Climada/climada_python/climada/util/lines_polys_handler.
↪py:931: UserWarning: Geometry is in a geographic CRS. Results from 'length' are␣
↪likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected␣
↪CRS before this operation.

  line_lengths = gdf_lines.length
```

```
u_lp.plot_eai_exp_geom(impact);
```




```
#disaggregate in meters
```

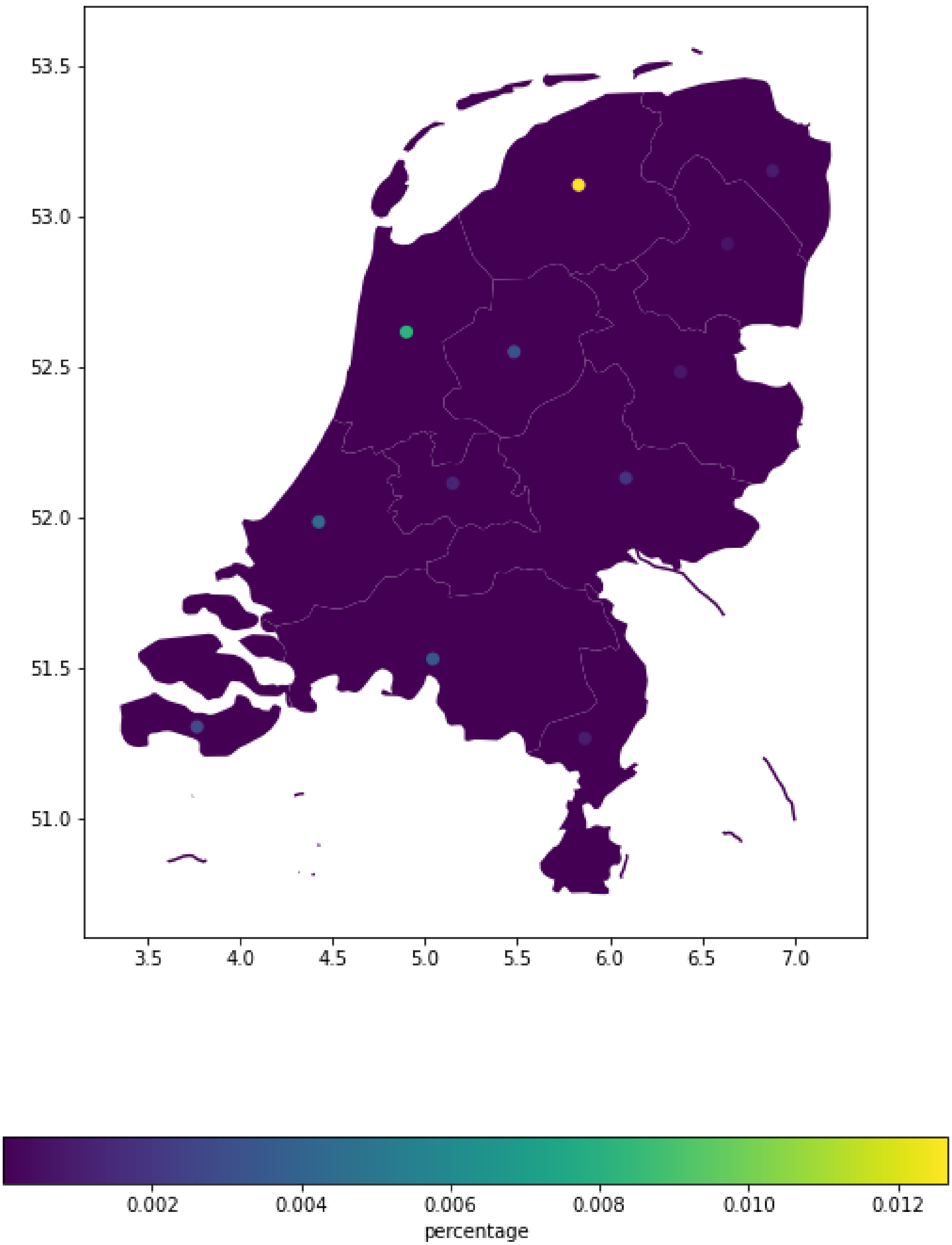
(continues on next page)

(continued from previous page)

```
#same value for each point, fixed to 1 (allows to get percentages of affected surface/
↪distance)
#aggregate by summing

impact = u_lp.calc_geom_impact(
            exp=EXP_MIX, impf_set=IMPF_SET, haz=HAZ,
            res=1000, to_meters=True, disagg_met=u_lp.DisaggMethod.FIX, disagg_val=1.
↪0,
            agg_met=u_lp.AggMethod.SUM
            );
```

```
2022-06-24 13:16:17,387 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:16:18,069 - climada.entity.exposures.base - INFO - Matching 37357␣
↪exposures with 9944 centroids.
2022-06-24 13:16:18,073 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:18,114 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:16:18,117 - climada.engine.impact - INFO - Calculating damage for 37357␣
↪assets (>0) and 2 events.
```

```
import matplotlib.pyplot as plt
ax = u_lp.plot_eai_exp_geom(impact, legend_kwds={'label': 'percentage', 'orientation
↪': 'horizontal'})
```


53.5
53.0
52.5
52.0
51.5
51.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
0.002
0.004
0.006
0.008
0.010
0.012
percentage

9.3.4 Polygons

Polygons or shapes are a common geographical representation of countries, states etc. as for example in NaturalEarth. Map data, as for example buildings, etc. obtained from openstreetmap (see tutorial here), also frequently come as (multi-)polygons. Here we want to show you how to deal with exposure information as polygons.

Load data

Lets assume we have the following data given. The polygons of the admin-1 regions of the Netherlands and an exposure value each, which we gather in a geodataframe. We want to know the Impact of Lothar on each admin-1 region.

In this tutorial, we shall see how to compute impacts for exposures defined on shapely geometries (polygons and/or lines). The basic principle is to disaggregate the geometries to a raster of points, compute the impact per points, and then re-aggregate. To do so, several methods are available. Here is a brief overview.

```
# Imports
import geopandas as gpd
import pandas as pd
from pathlib import Path

from climada.entity import Exposures
from climada.entity.impact_funcs.storm_europe import ImpfStormEurope
from climada.entity.impact_funcs import ImpactFuncSet
from climada.engine import ImpactCalc
from climada.hazard.storm_europe import StormEurope
import climada.util.lines_polys_handler as u_lp
from climada.util.constants import DEMO_DIR, WS_DEMO_NC
```

```
def gdf_poly():
    from cartopy.io import shapereader
    from climada_petals.entity.exposures.black_marble import country_iso_geom

    # open the file containing the Netherlands admin-1 polygons
    shp_file = shapereader.natural_earth(resolution='10m',
                                         category='cultural',
                                         name='admin_0_countries')
    shp_file = shapereader.Reader(shp_file)

    # extract the NL polygons
    prov_names = {'Netherlands': ['Groningen', 'Drenthe',
                              'Overijssel', 'Gelderland',
                                           'Limburg', 'Zeeland',
                                           'Noord-Brabant', 'Zuid-Holland',
                                           'Noord-Holland', 'Friesland',
                                           'Flevoland', 'Utrecht']
                                }
    polygon_Netherlands, polygons_prov_NL = country_iso_geom(prov_names,
                                shp_file)
    prov_geom_NL = {prov: geom for prov, geom in zip(list(prov_names.values())[0],
↪list(polygons_prov_NL.values())[0])}

    # assign a value to each admin-1 area (assumption 100'000 USD per inhabitant)
    population_prov_NL = {'Drenthe':493449, 'Flevoland':422202,
                     'Friesland':649988, 'Gelderland':2084478,
                     'Groningen':585881, 'Limburg':1118223,
                     'Noord-Brabant':2562566, 'Noord-Holland':2877909,
```

(continues on next page)

(continued from previous page)

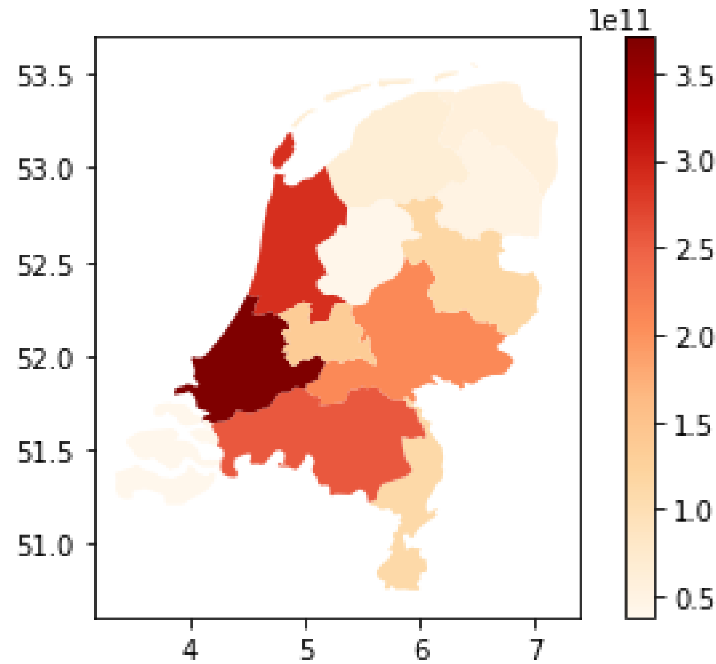
```
                        'Overijssel':1162215, 'Zuid-Holland':3705625,
                        'Utrecht':1353596, 'Zeeland':383689}
    value_prov_NL = {n: 100000 * population_prov_NL[n] for n in population_prov_NL.
↪keys()}

    # combine into GeoDataFrame and add a coordinate reference system to it:
    df1 = pd.DataFrame.from_dict(population_prov_NL, orient='index', columns=[
↪'population']).join(
          pd.DataFrame.from_dict(value_prov_NL, orient='index', columns=['value']))
    df1['geometry'] = [prov_geom_NL[prov] for prov in df1.index]
    gdf_polys = gpd.GeoDataFrame(df1)
    gdf_polys = gdf_polys.set_crs(epsg=4326)
    return gdf_polys
```

```
exp_nl_poly = Exposures(gdf_poly())
exp_nl_poly.gdf['impf_WS'] = 1
exp_nl_poly.gdf.head()
```

```
            population         value  \
Drenthe         493449   49344900000
Flevoland       422202   42220200000
Friesland       649988   64998800000
Gelderland     2084478  208447800000
Groningen       585881   58588100000

                                                      geometry  impf_WS
Drenthe     POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
Flevoland   POLYGON ((5.74046 52.83874, 5.75012 52.83507, ...        1
Friesland   MULTIPOLYGON (((5.17977 53.00899, 5.27947 53.0...        1
Gelderland  POLYGON ((6.77158 52.10879, 6.76587 52.10840, ...        1
Groningen   MULTIPOLYGON (((7.19459 53.24502, 7.19747 53.2...        1
```

```
# take a look
exp_nl_poly.gdf.plot('value', legend=True, cmap='OrRd')
```

```
<AxesSubplot:>
```




```
# define hazard
storms = StormEurope.from_footprints(WS_DEMO_NC)
# define impact function
impf = ImpfStormEurope.from_welker()
impf_set = ImpactFuncSet([impf])
```

```
2022-06-24 13:16:20,039 - climada.hazard.storm_europe - INFO - Constructing centroids␣
↪from /Users/ckropf/climada/demo/data/fp_lothar_crop-test.nc
2022-06-24 13:16:20,124 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
```

```
/Users/ckropf/Documents/Climada/climada_python/climada/hazard/centroids/centr.py:822:␣
↪UserWarning: Geometry is in a geographic CRS. Results from 'buffer' are likely␣
↪incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS␣
↪before this operation.

  xy_pixels = self.geometry.buffer(res / 2).envelope
```

```
2022-06-24 13:16:22,004 - climada.hazard.storm_europe - INFO - Commencing to iterate␣
↪over netCDF files.
```

Compute polygon impacts - all in one

All in one: The main method `calc_geom_impact` provides several disaggregation keywords, specifiying

- the target resolution (`res`),
- the method on how to distribute the values of the original geometries onto the newly generated interpolated points (`disagg_met`)
- the source (and number) of the value to be distributed (`disagg_val`).
- the aggregation method (`agg_met`)

disagg_met can be either fixed (FIX), replicating the original shape's value onto all points, or divided evenly (DIV), in which case the value is divided equally onto all new points. disagg_val can either be taken directly from the exposure gdf's value column (None) or be indicated here explicitly (float). Resolution can be given in the gdf's original (mostly degrees lat/lon) format, or in metres. agg_met can currently be only (SUM) were the value is summed over all points in the geometry.

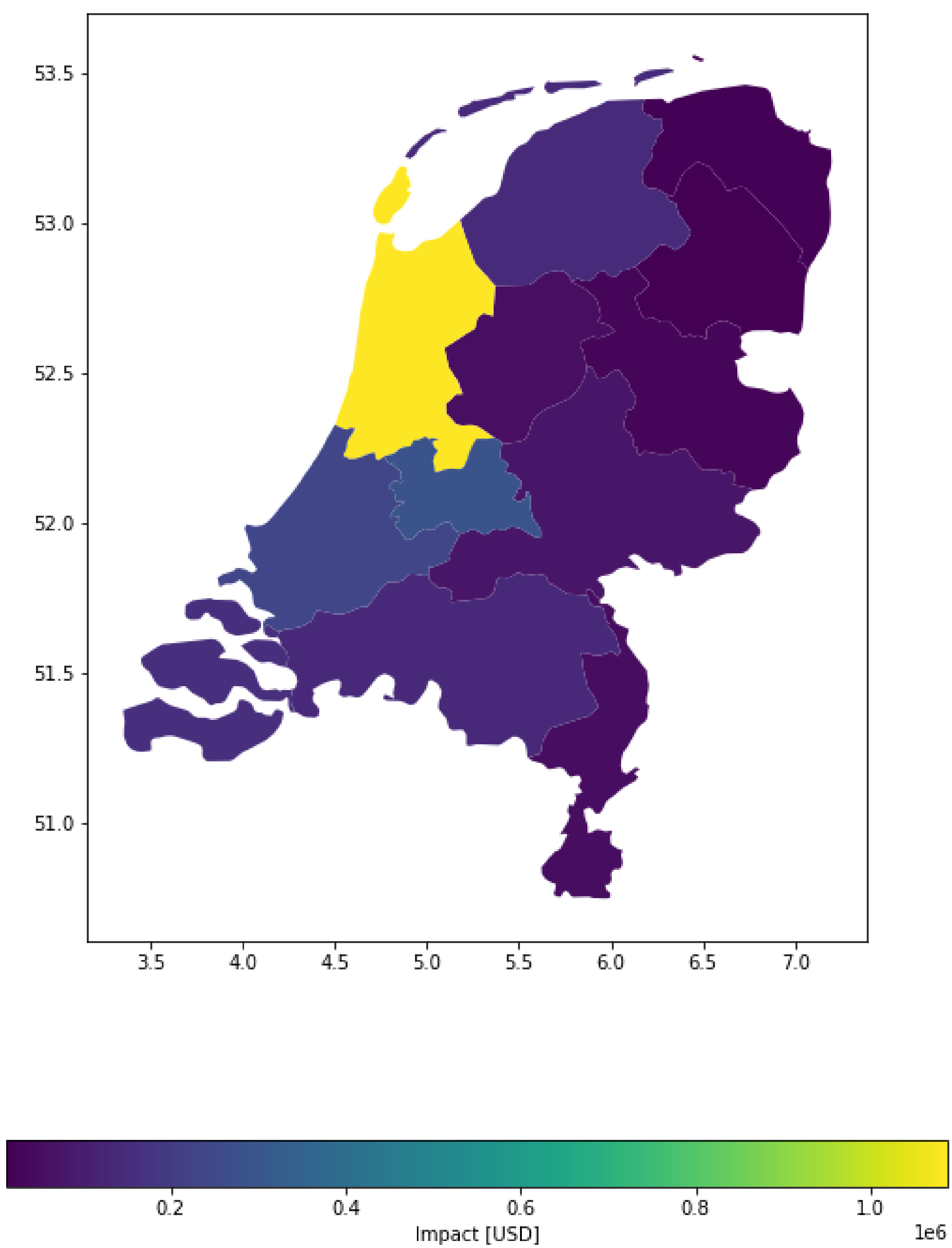
Polygons can also be disaggregated on a given fixed grid, see example below

Example 1: Target resolution in degrees lat/lon, equal (average) distribution of values from exposure gdf among points.

```
imp_deg = u_lp.calc_geom_impact(
    exp=exp_nl_poly, impf_set=impf_set, haz=storms,
    res=0.005, disagg_met=u_lp.DisaggMethod.DIV, disagg_val=None,
    agg_met=u_lp.AggMethod.SUM
)
```

```
2022-06-24 13:16:30,535 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:16:33,946 - climada.entity.exposures.base - INFO - Matching 195323␣
↪exposures with 9944 centroids.
2022-06-24 13:16:33,950 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:34,140 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:16:34,144 - climada.engine.impact - INFO - Calculating damage for␣
↪195323 assets (>0) and 2 events.
```

```
u_lp.plot_eai_exp_geom(imp_deg);
```

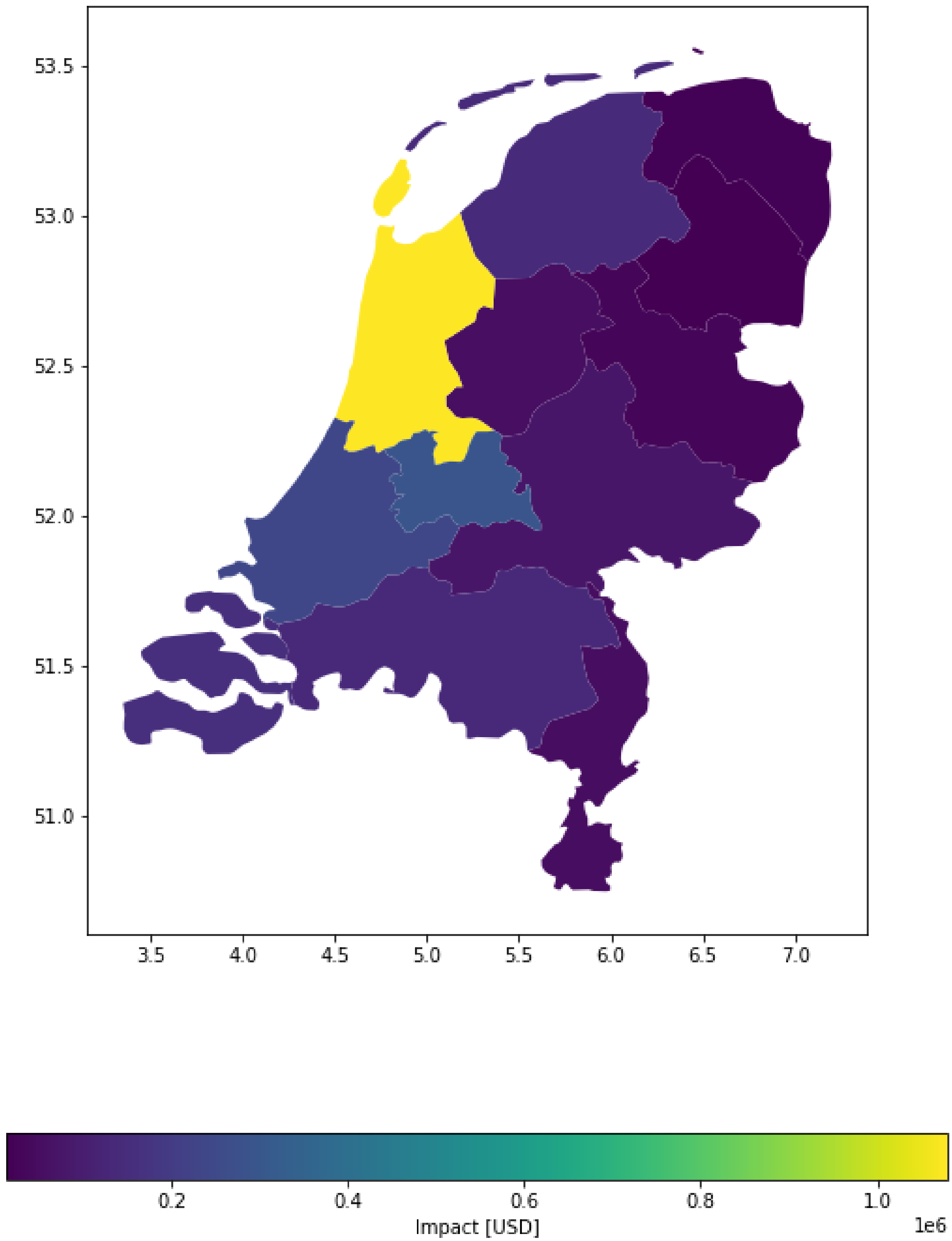


Example 2: Target resolution in metres, equal (divide) distribution of values from exposure gdf among points.

```
imp_m = u_lp.calc_geom_impact(
    exp=exp_nl_poly, impf_set=impf_set, haz=storms,
    res=500, to_meters=True, disagg_met=u_lp.DisaggMethod.DIV, disagg_val=None,
    agg_met=u_lp.AggMethod.SUM
)
```

```
2022-06-24 13:16:41,366 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:16:44,164 - climada.entity.exposures.base - INFO - Matching 148369␣
↪exposures with 9944 centroids.
2022-06-24 13:16:44,168 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:44,387 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:16:44,390 - climada.engine.impact - INFO - Calculating damage for␣
↪148369 assets (>0) and 2 events.
```

```
u_lp.plot_eai_exp_geom(imp_m);
```



For this specific case, both disaggregation methods provide a relatively similar result, given the chosen numbers:

```
(imp_deg.eai_exp - imp_m.eai_exp) / imp_deg.eai_exp
```

```
array([ 0.00614586, -0.01079377,  0.00021355,  0.00140608,  0.0038771 ,
       -0.0066888 , -0.00171755,  0.00741871, -0.00107029,  0.00424221,
       -0.01838225,  0.01811858])
```

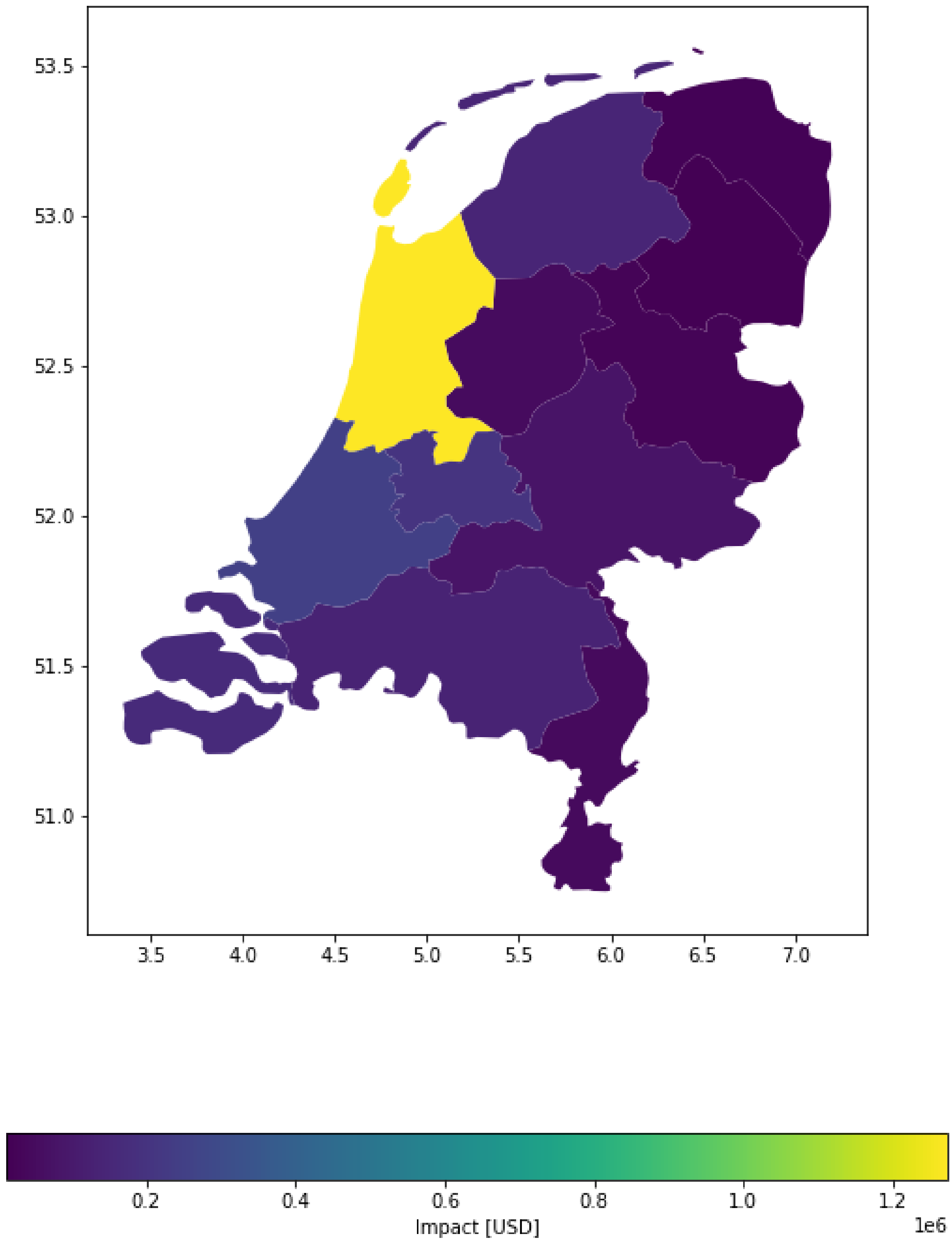
Example 3: Target predefined grid, equal (divide) distribution of values from exposure gdf among points.

```
#regular grid from exposures bounds
import climada.util.coordinates as u_coord
res = 0.1
(_, _, xmax, ymax) = exp_nl_poly.gdf.geometry.bounds.max()
(xmin, ymin, _, _) = exp_nl_poly.gdf.geometry.bounds.min()
bounds = (xmin, ymin, xmax, ymax)
height, width, trafo = u_coord.pts_to_raster_meta(
    bounds, (res, res)
    )
x_grid, y_grid = u_coord.raster_to_meshgrid(trafo, width, height)
```

```
imp_g = u_lp.calc_grid_impact(
    exp=exp_nl_poly, impf_set=impf_set, haz=storms,
    grid=(x_grid, y_grid), disagg_met=u_lp.DisaggMethod.DIV, disagg_val=None,
    agg_met=u_lp.AggMethod.SUM
)
```

```
2022-06-24 13:16:45,086 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:16:45,100 - climada.entity.exposures.base - INFO - Matching 486␣
↪exposures with 9944 centroids.
2022-06-24 13:16:45,102 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:45,110 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:16:45,111 - climada.engine.impact - INFO - Calculating damage for 486␣
↪assets (>0) and 2 events.
```

```
u_lp.plot_eai_exp_geom(imp_g);
```


53.5
53.0
52.5
52.0
51.5
51.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
0.2
0.4
0.6
0.8
1.0
1.2
Impact [USD]
1e6

Compute polygon impacts - step by step

Step 1: Disaggregate polygon exposures to points. It is useful to do this separately, when the discretized exposure is used several times, for example, to compute with different hazards.

Several disaggregation methods can be used as shown below:

```
# Disaggregate exposure to 10'000 metre grid, each point gets average value within␣
↪polygon.
exp_pnt = u_lp.exp_geom_to_pnt(
    exp_nl_poly, res=10000, to_meters=True,
    disagg_met=u_lp.DisaggMethod.DIV, disagg_val=None
)
exp_pnt.gdf.head()
```

```
2022-06-24 13:16:45,445 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
           population         value  \
Drenthe 0      493449  2.056038e+09
        1      493449  2.056038e+09
        2      493449  2.056038e+09
        3      493449  2.056038e+09
        4      493449  2.056038e+09

                                               geometry_orig  impf_WS  \
Drenthe 0  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        1  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        2  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        3  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        4  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1

                           geometry   latitude  longitude
Drenthe 0  POINT (6.21931 52.75939)  52.759394   6.219308
        1  POINT (6.30914 52.75939)  52.759394   6.309139
        2  POINT (6.39897 52.75939)  52.759394   6.398971
        3  POINT (6.48880 52.75939)  52.759394   6.488802
        4  POINT (6.57863 52.75939)  52.759394   6.578634
```

```
# Disaggregate exposure to 0.1° grid, no value disaggregation specified --> replicate␣
↪initial value
exp_pnt2 = u_lp.exp_geom_to_pnt(
    exp_nl_poly, res=0.1, to_meters=False,
    disagg_met=u_lp.DisaggMethod.FIX, disagg_val=None)
exp_pnt2.gdf.head()
```

```
2022-06-24 13:16:45,516 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
           population        value  \
Drenthe 0      493449  49344900000
        1      493449  49344900000
        2      493449  49344900000
        3      493449  49344900000
        4      493449  49344900000
```

(continues on next page)

(continued from previous page)

```
                                                  geometry_orig  impf_WS  \
Drenthe 0  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        1  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        2  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        3  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        4  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1

                            geometry   latitude  longitude
Drenthe 0  POINT (6.22948 52.71147)  52.711466   6.229476
        1  POINT (6.32948 52.71147)  52.711466   6.329476
        2  POINT (6.42948 52.71147)  52.711466   6.429476
        3  POINT (6.52948 52.71147)  52.711466   6.529476
        4  POINT (6.62948 52.71147)  52.711466   6.629476
```

```
# Disaggregate exposure to 1'000 metre grid, each point gets value corresponding to
# its representative area (1'000^2).
exp_pnt3 = u_lp.exp_geom_to_pnt(
    exp_nl_poly, res=1000, to_meters=True,
    disagg_met=u_lp.DisaggMethod.FIX, disagg_val=10e6)
exp_pnt3.gdf.head()
```

```
2022-06-24 13:16:47,258 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
           population       value  \
Drenthe 0      493449  10000000.0
        1      493449  10000000.0
        2      493449  10000000.0
        3      493449  10000000.0
        4      493449  10000000.0

                                                  geometry_orig  impf_WS  \
Drenthe 0  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        1  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        2  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        3  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        4  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1

                            geometry   latitude  longitude
Drenthe 0  POINT (6.38100 52.62624)  52.626236   6.381005
        1  POINT (6.38999 52.62624)  52.626236   6.389988
        2  POINT (6.39897 52.62624)  52.626236   6.398971
        3  POINT (6.40795 52.62624)  52.626236   6.407954
        4  POINT (6.41694 52.62624)  52.626236   6.416937
```

```
# Disaggregate exposure to 1'000 metre grid, each point gets value corresponding to 1
# After dissagregation, each point has a value equal to the percentage of area of the␣
↪polygon
exp_pnt4 = u_lp.exp_geom_to_pnt(
    exp_nl_poly, res=1000, to_meters=True,
    disagg_met=u_lp.DisaggMethod.DIV, disagg_val=1)
exp_pnt4.gdf.tail()
```

```
2022-06-24 13:16:49,929 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
              population     value  \
Zeeland 1897      383689  0.000526
        1898      383689  0.000526
        1899      383689  0.000526
        1900      383689  0.000526
        1901      383689  0.000526

                                                   geometry_orig  impf_WS  \
Zeeland 1897  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        1898  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        1899  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        1900  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        1901  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1

                              geometry   latitude  longitude
Zeeland 1897  POINT (3.92434 51.73813)  51.738126   3.924336
        1898  POINT (3.93332 51.73813)  51.738126   3.933320
        1899  POINT (3.94230 51.73813)  51.738126   3.942303
        1900  POINT (3.95129 51.73813)  51.738126   3.951286
        1901  POINT (3.96027 51.73813)  51.738126   3.960269
```

```
# disaggregate on pre-defined grid
#regular grid from exposures bounds
import climada.util.coordinates as u_coord
res = 0.1
(_, _, xmax, ymax) = exp_nl_poly.gdf.geometry.bounds.max()
(xmin, ymin, _, _) = exp_nl_poly.gdf.geometry.bounds.min()
bounds = (xmin, ymin, xmax, ymax)
height, width, trafo = u_coord.pts_to_raster_meta(
    bounds, (res, res)
    )
x_grid, y_grid = u_coord.raster_to_meshgrid(trafo, width, height)
exp_pnt5 = u_lp.exp_geom_to_grid(
    exp_nl_poly, grid=(x_grid, y_grid),
    disagg_met=u_lp.DisaggMethod.DIV, disagg_val=1)
exp_pnt5.gdf.tail()
```

```
2022-06-24 13:16:50,765 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
            population     value  \
Zeeland 17      383689  0.045455
        18      383689  0.045455
        19      383689  0.045455
        20      383689  0.045455
        21      383689  0.045455

                                                 geometry_orig  impf_WS  \
Zeeland 17  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        18  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        19  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        20  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        21  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1

                            geometry  latitude  longitude
Zeeland 17  POINT (3.84941 51.54755)  51.54755   3.849415
```

(continues on next page)

(continued from previous page)

```
        18  POINT (4.14941 51.54755)  51.54755   4.149415
        19  POINT (3.94941 51.64755)  51.64755   3.949415
        20  POINT (4.04941 51.64755)  51.64755   4.049415
        21  POINT (4.14941 51.64755)  51.64755   4.149415
```

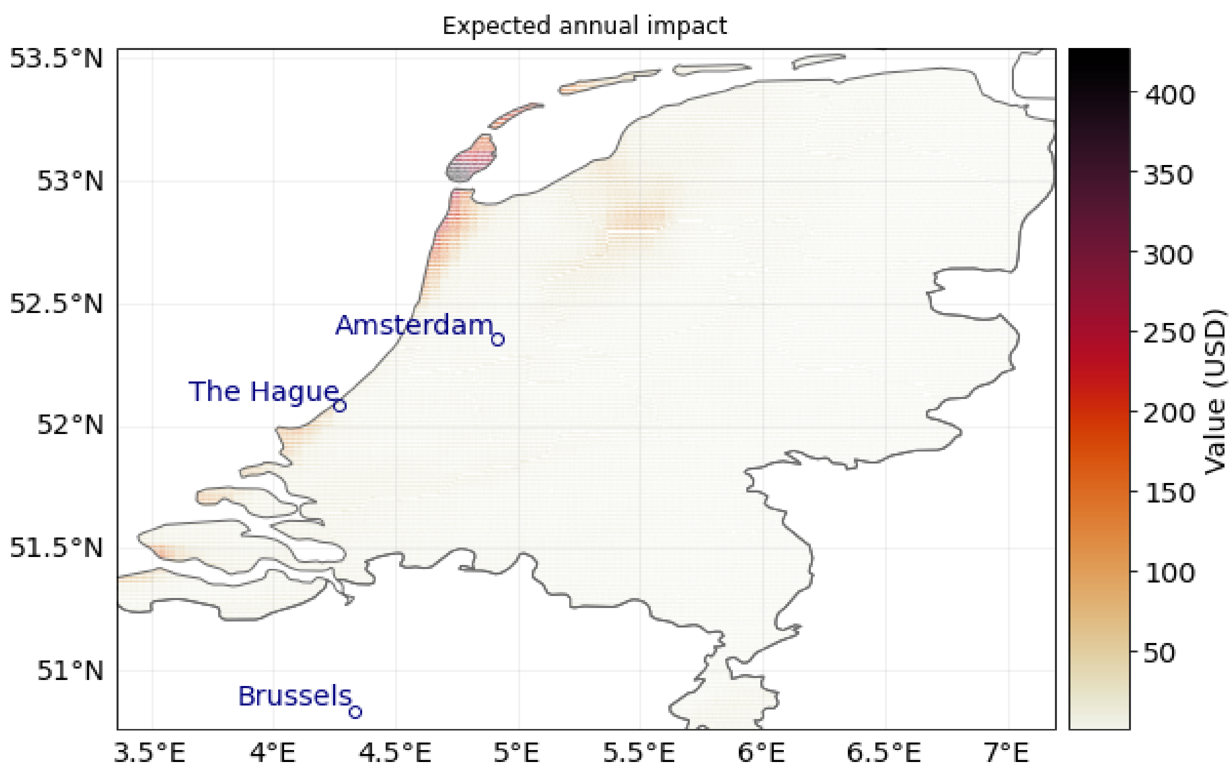
Step 2: Calculate point impacts & re-aggregate them afterwards

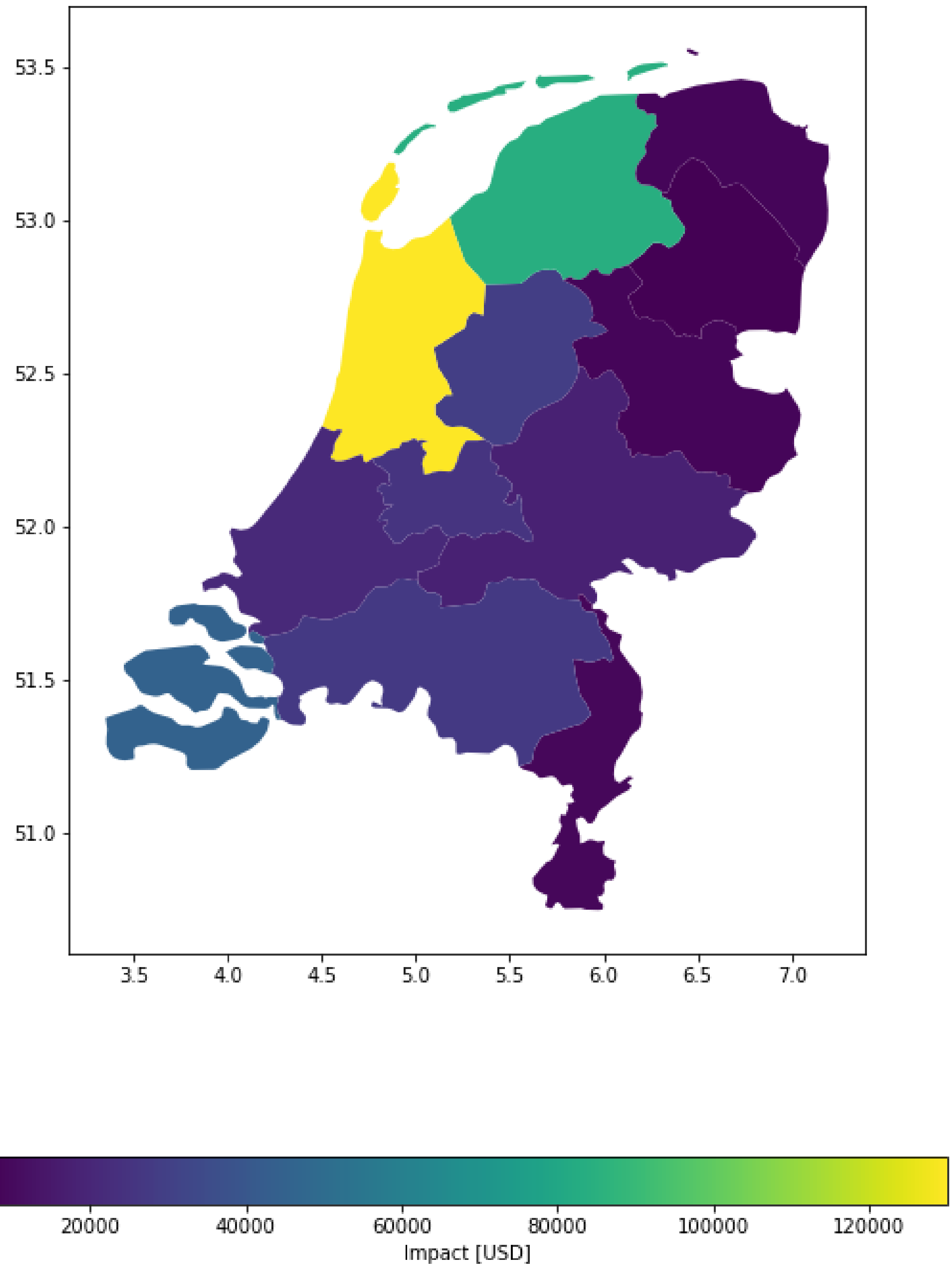
```
# Point-impact
imp_pnt = ImpactCalc(exp_pnt3, impf_set, hazard=storms).impact(save_mat=True)

# Aggregated impact (Note that you need to pass the gdf and not the exposures)
imp_geom = u_lp.impact_pnt_agg(imp_pnt, exp_pnt3.gdf, agg_met=u_lp.AggMethod.SUM)
```

```
2022-06-24 13:16:50,793 - climada.entity.exposures.base - INFO - Matching 37082␣
↪exposures with 9944 centroids.
2022-06-24 13:16:50,796 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:50,929 - climada.engine.impact - INFO - Calculating damage for 37082␣
↪assets (>0) and 2 events.
```

```
# Plot point-impacts and aggregated impacts
imp_pnt.plot_hexbin_eai_exposure();
u_lp.plot_eai_exp_geom(imp_geom);
```

```
2022-06-24 13:16:51,002 - climada.util.plot - WARNING - Error parsing coordinate␣
↪system 'epsg:4326'. Using projection PlateCarree in plot.
```





Aggregated impact, in detail

```
# aggregate impact matrix
mat_agg = u_lp._aggregate_impact_mat(imp_pnt, exp_pnt3.gdf, agg_met=u_lp.AggMethod.
↪SUM)

eai_exp = u_lp.eai_exp_from_mat(imp_mat=mat_agg, freq=imp_pnt.frequency)
at_event = u_lp.at_event_from_mat(imp_mat=mat_agg)
aai_agg = u_lp.aai_agg_from_at_event(at_event=at_event, freq=imp_pnt.frequency)
```

```
eai_exp
```

```
array([  6753.45186608,  28953.21638482,  84171.28903387,  18014.15630989,
         8561.18507994,   8986.13653385,  27446.46387061, 130145.29903078,
         8362.17243334,  20822.87844894,  25495.46296087,  45121.14833362])
```

```
at_event
```

```
array([4321211.03400214,  219950.4291506 ])
```

```
aai_agg
```

```
412832.8602866131
```

9.3.5 Lines

Lines are common geographical representation of transport infrastructure like streets, train tracks or powerlines etc. Here we will play it through for the case of winter storm Lothar's impact on the Dutch Railway System:

Loading Data

Note: Hazard and impact functions data have been loaded above.

```
def gdf_lines():
    gdf_lines = gpd.read_file(Path(DEMO_DIR,'nl_rails.gpkg'))
    gdf_lines = gdf_lines.to_crs(epsg=4326)
    return gdf_lines
```

```
exp_nl_lines = Exposures(gdf_lines())
exp_nl_lines.gdf['impf_WS'] = 1
exp_nl_lines.gdf['value'] = 1
exp_nl_lines.gdf.head()
```

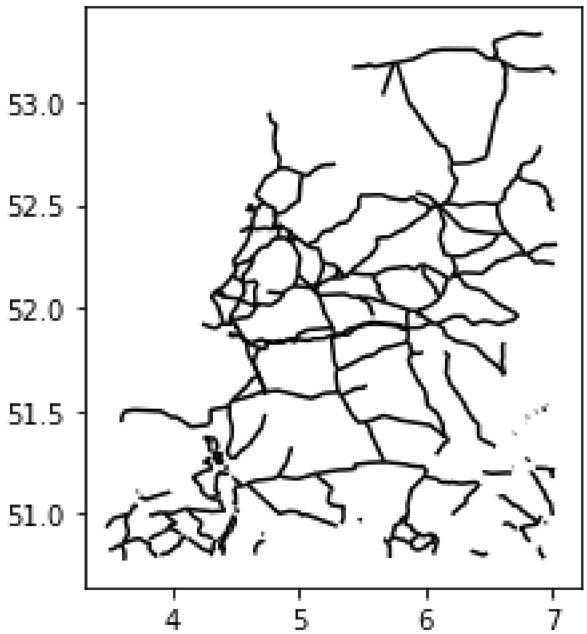
```
       distance                                           geometry  impf_WS  \
0   9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
1  10397.965112  LINESTRING (6.17673 51.35530, 6.17577 51.35410...        1
2  26350.219724  LINESTRING (5.68096 51.25040, 5.68422 51.25020...        1
3  40665.249638  LINESTRING (5.68096 51.25040, 5.67711 51.25030...        1
4   8297.689753  LINESTRING (5.70374 50.85490, 5.70531 50.84990...        1

   value
0      1
1      1
```

(continues on next page)

(continued from previous page)

```
2       1
3       1
4       1
```

```
exp_nl_lines.gdf.plot('value', cmap='inferno');
```



Calculating line impacts - all in one

Example 1: Disaggregate values evenly among road segments; split in points with 0.005 degree distances.

```
imp_deg = u_lp.calc_geom_impact(
    exp=exp_nl_lines, impf_set=impf_set, haz=storms,
    res=0.005, disagg_met=u_lp.DisaggMethod.DIV, disagg_val=None,
    agg_met=u_lp.AggMethod.SUM
)
```

```
/Users/ckropf/Documents/Climada/climada_python/climada/util/lines_polys_handler.
↪py:931: UserWarning: Geometry is in a geographic CRS. Results from 'length' are␣
↪likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected␣
↪CRS before this operation.

  line_lengths = gdf_lines.length
```

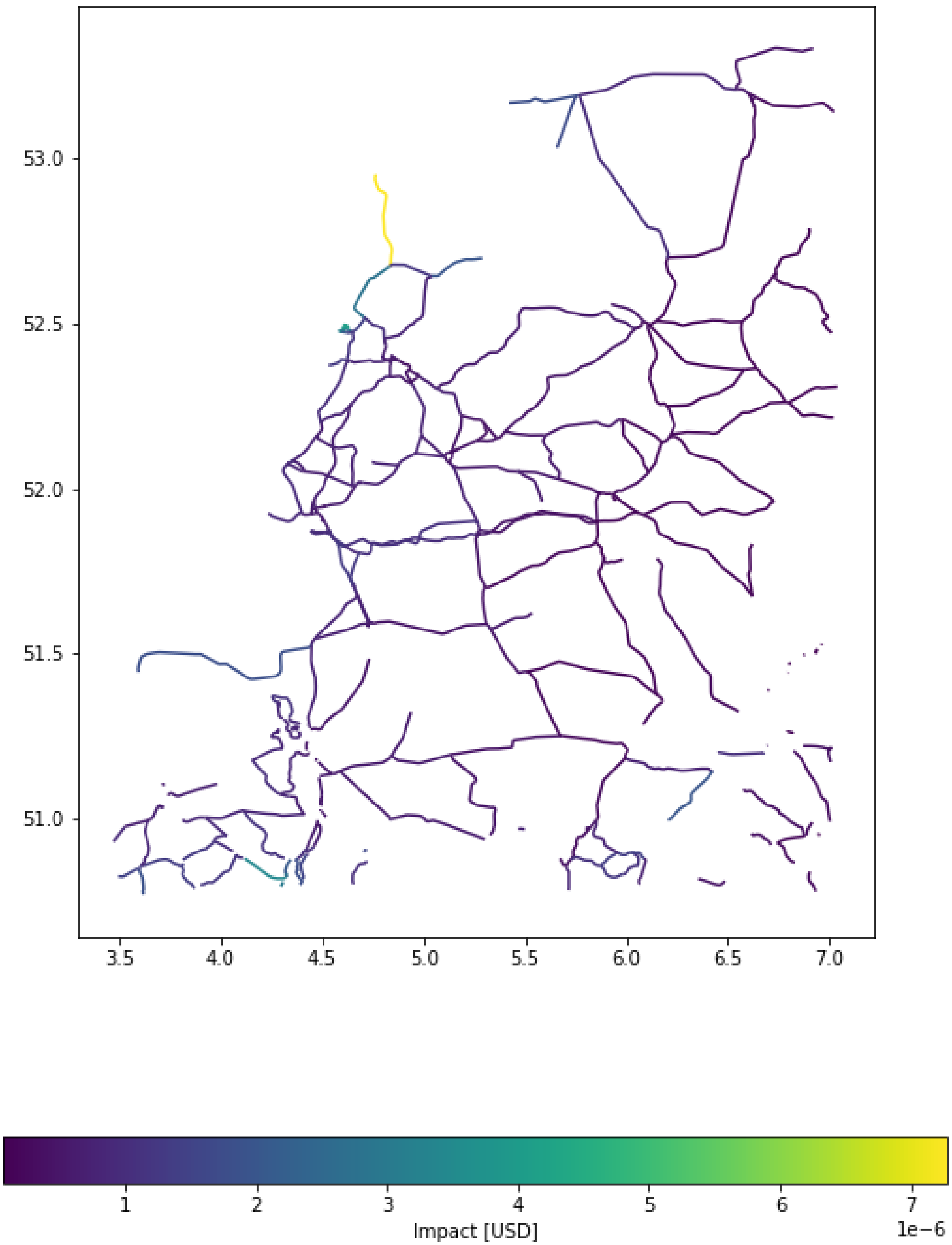
```
2022-06-24 13:16:59,608 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:16:59,787 - climada.entity.exposures.base - INFO - Matching 10175␣
↪exposures with 9944 centroids.
2022-06-24 13:16:59,789 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:59,805 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
```

(continues on next page)

(continued from previous page)

```
2022-06-24 13:16:59,806 - climada.engine.impact - INFO - Calculating damage for 10175␣
↪assets (>0) and 2 events.
```

```
u_lp.plot_eai_exp_geom(imp_deg);
```

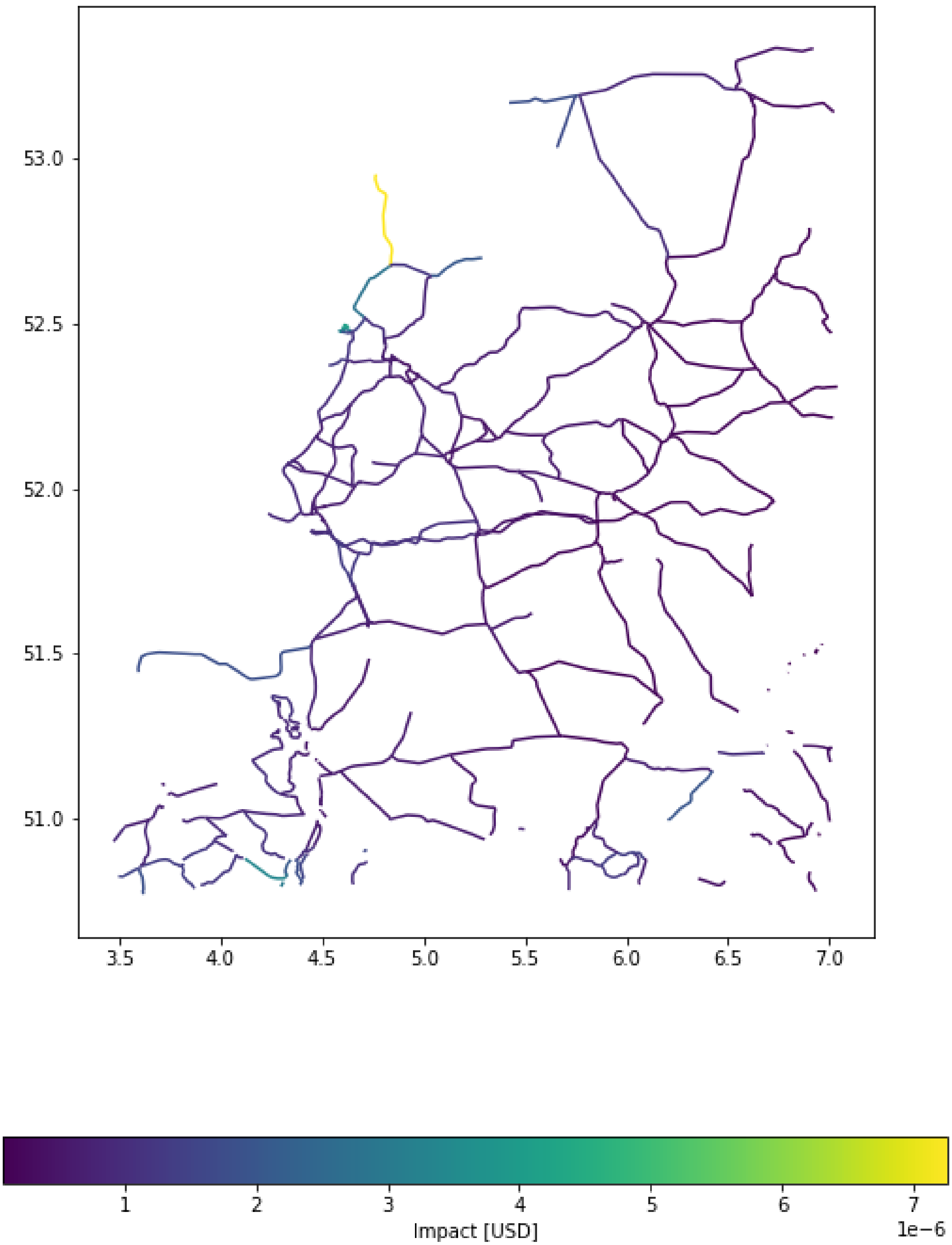


Example 2: Disaggregate values evenly among road segments; split in points with 500 m distances.

```
imp_m = u_lp.calc_geom_impact(
    exp=exp_nl_lines, impf_set=impf_set, haz=storms,
    res=500, to_meters=True, disagg_met=u_lp.DisaggMethod.DIV, disagg_val=None,
    agg_met=u_lp.AggMethod.SUM
)
```

```
2022-06-24 13:17:00,670 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:17:00,827 - climada.entity.exposures.base - INFO - Matching 8399␣
↪exposures with 9944 centroids.
2022-06-24 13:17:00,828 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:17:00,843 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:17:00,845 - climada.engine.impact - INFO - Calculating damage for 8399␣
↪assets (>0) and 2 events.
```

```
u_lp.plot_eai_exp_geom(imp_m);
```




```
import numpy as np
```

(continues on next page)

(continued from previous page)

```
diff = np.max((imp_deg.eai_exp - imp_m.eai_exp) / imp_deg.eai_exp)
print(f"The largest relative different between degrees and meters impact in this␣
↪example is {diff}")
```

```
The largest relative different between degrees and meters impact in this example is 0.
↪09803913811822067
```

Calculating line impacts - step by step

Step 1: As in the polygon example above, there are several methods to disaggregate line exposures into point exposures, of which several are shown here:

```
# 0.1° distance between points, average value disaggregation
exp_pnt = u_lp.exp_geom_to_pnt(
    exp_nl_lines, res=0.1, to_meters=False, disagg_met=u_lp.DisaggMethod.DIV, disagg_
↪val=None
)
exp_pnt.gdf.head()
```

```
2022-06-24 13:17:01,409 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
/Users/ckropf/Documents/Climada/climada_python/climada/util/lines_polys_handler.
↪py:931: UserWarning: Geometry is in a geographic CRS. Results from 'length' are␣
↪likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected␣
↪CRS before this operation.

  line_lengths = gdf_lines.length
```

```
          distance                                     geometry_orig  impf_WS  \
0 0    9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  1    9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
1 0   10397.965112  LINESTRING (6.17673 51.35530, 6.17577 51.35410...        1
  1   10397.965112  LINESTRING (6.17673 51.35530, 6.17577 51.35410...        1
  2   10397.965112  LINESTRING (6.17673 51.35530, 6.17577 51.35410...        1

        value                  geometry   latitude  longitude
0 0  0.500000  POINT (6.08850 50.87940)  50.879400   6.088500
  1  0.500000  POINT (6.06079 50.80030)  50.800300   6.060790
1 0  0.333333  POINT (6.17673 51.35530)  51.355300   6.176730
  1  0.333333  POINT (6.12632 51.32440)  51.324399   6.126323
  2  0.333333  POINT (6.08167 51.28460)  51.284600   6.081670
```

```
# 1000m distance between points, no value disaggregation
exp_pnt2 = u_lp.exp_geom_to_pnt(
    exp_nl_lines, res=1000, to_meters=True, disagg_met=u_lp.DisaggMethod.FIX, disagg_
↪val=None
)
exp_pnt2.gdf.head()
```

```
2022-06-24 13:17:01,876 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
        distance                                      geometry_orig  impf_WS  \
0 0  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  1  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  2  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  3  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  4  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1

     value                  geometry   latitude  longitude
0 0      1  POINT (6.08850 50.87940)  50.879400   6.088500
  1      1  POINT (6.09416 50.87275)  50.872755   6.094165
  2      1  POINT (6.09161 50.86410)  50.864105   6.091608
  3      1  POINT (6.08744 50.85590)  50.855902   6.087435
  4      1  POINT (6.08326 50.84770)  50.847699   6.083263
```

```
# 1000m distance between points, equal value disaggregation
exp_pnt3 = u_lp.exp_geom_to_pnt(
    exp_nl_lines, res=1000, to_meters=True, disagg_met=u_lp.DisaggMethod.DIV, disagg_
↪val=None
)
exp_pnt3.gdf.head()
```

```
2022-06-24 13:17:02,436 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
        distance                                      geometry_orig  impf_WS  \
0 0  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  1  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  2  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  3  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  4  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1

        value                  geometry   latitude  longitude
0 0  0.090909  POINT (6.08850 50.87940)  50.879400   6.088500
  1  0.090909  POINT (6.09416 50.87275)  50.872755   6.094165
  2  0.090909  POINT (6.09161 50.86410)  50.864105   6.091608
  3  0.090909  POINT (6.08744 50.85590)  50.855902   6.087435
  4  0.090909  POINT (6.08326 50.84770)  50.847699   6.083263
```

```
# 1000m distance between points, disaggregation of value according to representative␣
↪distance
exp_pnt4 = u_lp.exp_geom_to_pnt(
    exp_nl_lines, res=1000, to_meters=True,  disagg_met=u_lp.DisaggMethod.FIX, disagg_
↪val=1000
)
exp_pnt4.gdf.head()
```

```
2022-06-24 13:17:03,116 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
        distance                                      geometry_orig  impf_WS  \
0 0  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  1  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  2  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  3  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  4  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
```

(continues on next page)

(continued from previous page)

```
     value                  geometry   latitude  longitude
0 0   1000  POINT (6.08850 50.87940)  50.879400   6.088500
  1   1000  POINT (6.09416 50.87275)  50.872755   6.094165
  2   1000  POINT (6.09161 50.86410)  50.864105   6.091608
  3   1000  POINT (6.08744 50.85590)  50.855902   6.087435
  4   1000  POINT (6.08326 50.84770)  50.847699   6.083263
```

Step 2 & 3: The procedure is analogous to the example provided above for polygons.

CHAPTER

TEN

IMPACT TUTORIALS

10.1 END-TO-END IMPACT CALCULATION

10.1.1 Goal of this tutorial

The goal of this tutorial is to show a full end-to-end impact computation. Note that this tutorial exemplifies the work flow, but does not explore all possible features.

10.1.2 What is an Impact?

The impact is the combined effect of hazard events on a set of exposures mediated by a set of impact functions. By computing the impact for each event (historical and synthetic) and for each exposure value at each geographical location, the Impact class provides different risk measures, such as the expected annual impact per exposure, the probable maximum impact for different return periods, and the total average annual impact.

10.1.3 Impact class data structure

The impact class does not require any attributes to be defined by the user. All attributes are set by the method `ImpactCalc.impact()`. This method requires three attributes: an `Exposure`, a `Hazard`, and an `ImpactFuncSet`. After calling `ImpactCalc(Exposure, ImpactFuncSet, Hazard).impact(save_mat=False)`, the Impact object has the following attributes:

Attributes from input	Data Type	Description
event_id	list(int)	id (>0) of each hazard event (Hazard.event_id)
event_name	(list(str))	name of each event (Hazard.event_name)
date	np.array	date of events (Hazard.date)
coord_exp	np.array	exposures coordinates [lat, lon] (in degrees) (Exposure.gdf.latidues, Exposure.gdf.longitude)
frequency	np.array	frequency of events (Hazard.frequency)
frequency_unit	str	unit of event frequency, by default '1/year', i.e., annual (Hazard.frequency_unit)
unit	str	value unit used (Exposure.value_unit)
csr	str	unit system for Exposure and Hazard geographical data (Exposure.csr)

Computed attributes	Data Type	Description
at_event	np.array	impact for each hazard event summed over all locations
eai_exp	np.array	expected annual impact for each locations, summed over all events weigthed by frequency
aai_agg	float	total annual average aggregated impact value (summed over events and locations)
impt_mat	sparse.csr_	This is the impact matrix. This matrix has the events as rows and the exposure points as columns (num_events X num_exp). It will only be filled with impact values if save_mat is True.
tot_value	float	total exposure value affected (sum of value all exposures locations affected by at least one hazard event)

All other methods compute values from the attributes set by `ImpactCalc.impact()`. For example, one can compute the frequency exceedance curve, plot impact data, or compute traditional risk transfer over impact.

10.1.4 How do I compute an impact in CLIMADA?

In CLIMADA, impacts are computed using the `Impact` class. The computation of the impact requires an `Exposure` , an `ImpactFuncSet`, and a `Hazard` object. For details about how to define ***Exposures*** , ***Hazard***, ***Impact Functions*** see the respective tutorials.

The steps of an impact caculations are typically:

- Set exposure
- Set hazard and hazard centroids
- Set impact functions in impact function set
- Compute impact
- Visualize, save, use impact output

Hints: Before computing the impact of a given `Exposure` and `Hazard`, it is important to correctly match the `Exposures` coordinates with the `Hazard Centroids`. Try to have similar resolutions in `Exposures` and `Hazard`. By the impact calculation the nearest neighbor for each `Exposure` to the `Hazard's Centroids` is searched.

Hint: Set first the `Exposures` and use its coordinates information to set a matching `Hazard`.

Hint: The configuration value `max_matrix_size` controls the maximum matrix size contained in a chunk. By default it is set to 1e9 in the default config file. A high value makes the computation fast at the cost of increased memory consumption. You can decrease its value if you are having memory issues with the `ImpactCalc.impact()` method. (See the *config guide* on how to set configuration values).

10.1.5 Structure of the tutorial

We begin with one very detailled example, and later present in quick and dirty examples.

Part1: Detailed impact calculation with Litpop and TropCyclone

Part2: Quick examples: raster and point exposures/hazards

Part3: Visualization methods

10.1.6 Detailed Impact calculation - LitPop + TropCyclone

We present a detailed example for the hazard ***Tropical Cyclones*** and the exposures from ***LitPop*** .

Define the exposure

Reminder: The exposures must be defined according to your problem either using CLIMADA exposures such as **BlackMarble**, ***LitPop***, **OSM**, extracted from external sources (imported via csv, excel, api, …) or directly user defined.

As a reminder, exposures are geopandas dataframes with at least columns 'latitude', 'longitude' and 'value' of exposures.

For impact calculations, for each exposure values of the corresponding impact function to use (defined by the column `impf_`) and the associated hazard centroids must be defined. This is done after defining the impact function(s) and the hazard(s).

See tutorials on ***Exposures*** , ***Hazard***, ***ImpactFuncSet*** for more details.

Exposures are either defined as a series of (latitude/longitude) points or as a raster of (latitude/longitude) points. Fundamentally, this changes nothing for the impact computations. Note that for larger number of points, consider using a raster which might be more efficient (computationally). For a low number of points, avoid using a raster if this adds a lot of exposures values equal to 0.

We shall here use a raster example.

```
# Exposure from the module Litpop
# Note that the file gpw_v4_population_count_rev11_2015_30_sec.tif must be downloaded␣
↪(do not forget to unzip) if
# you want to execute this cell on your computer. If you haven't downloaded it before,
↪ please have a look at the section
# "population count" of the LitPop tutorial.

%matplotlib inline
import numpy as np
from climada.entity import LitPop

# Cuba with resolution 10km and financial_mode = income group.
exp_lp = LitPop.from_countries(countries=['CUB'], res_arcsec=300, fin_mode='income_
↪group')
exp_lp.check()
```

```
2023-01-26 11:57:14,980 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2023-01-26 11:57:15,071 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,073 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,130 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,132 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,193 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,195 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,305 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
```

(continues on next page)

(continued from previous page)

```
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,307 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,371 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,373 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,410 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:15,412 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,413 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,444 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:15,445 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,448 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,488 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:15,488 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,491 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,521 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:15,523 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,525 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,603 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,605 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,661 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,663 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,724 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,725 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,791 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,792 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,826 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:15,828 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,831 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,892 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,894 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2023-01-26 11:57:15,958 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,959 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:16,033 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:16,035 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:16,771 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:16,773 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:16,831 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:16,833 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:16,866 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:16,868 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:16,871 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:16,939 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:16,941 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,005 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,007 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,066 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,067 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,127 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,130 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,163 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:17,166 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,169 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,197 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:17,199 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,202 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,269 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,271 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,334 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,336 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
```

```
2023-01-26 11:57:17,411 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,413 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,445 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:17,448 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,451 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,476 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:17,477 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,479 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,546 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,550 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,582 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:17,584 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,585 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,645 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,646 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,702 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,704 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,769 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,771 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,837 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,840 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,924 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,925 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,998 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:18,000 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:18,066 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:18,068 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

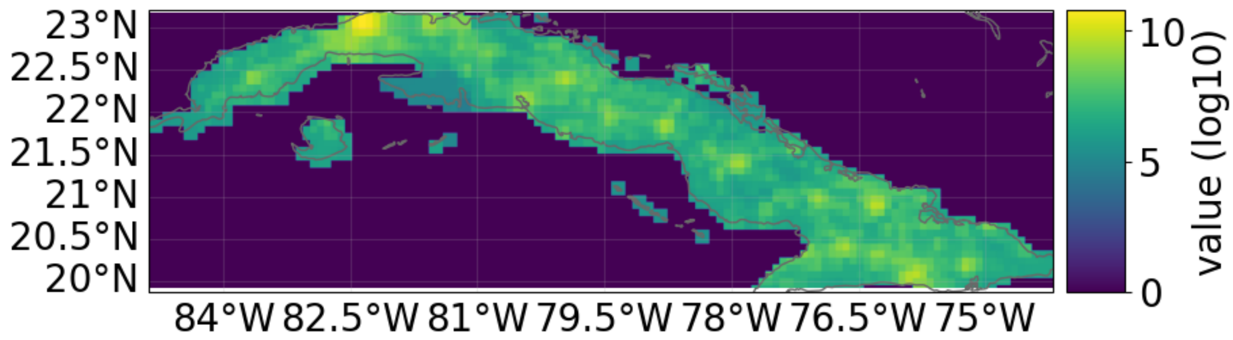
```
2023-01-26 11:57:18,100 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:18,576 - climada.util.finance - INFO - GDP CUB 2018: 1.000e+11.
2023-01-26 11:57:18,671 - climada.util.finance - INFO - Income group CUB 2018: 3.
2023-01-26 11:57:18,692 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-01-26 11:57:18,694 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:57:18,695 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:57:18,697 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:57:18,699 - climada.entity.exposures.base - INFO - centr_ not set.
2023-01-26 11:57:18,700 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-01-26 11:57:18,702 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:57:18,703 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:57:18,704 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:57:18,706 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
exp_lp.gdf.head()
```

```
          value                    geometry   latitude  longitude  region_id  \
0  1.077368e+05  POINT (-81.37500 21.70833)  21.708333 -81.375000        192
1  1.671873e+06  POINT (-81.54167 21.62500)  21.625000 -81.541667        192
2  3.421208e+06  POINT (-82.95833 21.87500)  21.875000 -82.958333        192
3  1.546590e+07  POINT (-82.87500 21.87500)  21.875000 -82.875000        192
4  7.168305e+07  POINT (-82.79167 21.87500)  21.875000 -82.791667        192

   impf_
0      1
1      1
2      1
3      1
4      1
```

```
# not needed for impact calculations
# visualize the define exposure
exp_lp.plot_raster()
print('\n Raster properties exposures:', exp_lp.meta)
```

```
2023-01-26 11:57:18,777 - climada.util.coordinates - INFO - Raster from resolution 0.
↪08333332999999854 to 0.08333332999999854.

 Raster properties exposures: {'width': 129, 'height': 41, 'crs': <Geographic 2D CRS:␣
↪EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- undefined
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
, 'transform': Affine(0.08333333000000209, 0.0, -84.91666666500001,
       0.0, -0.08333332999999854, 23.249999994999996)}
```



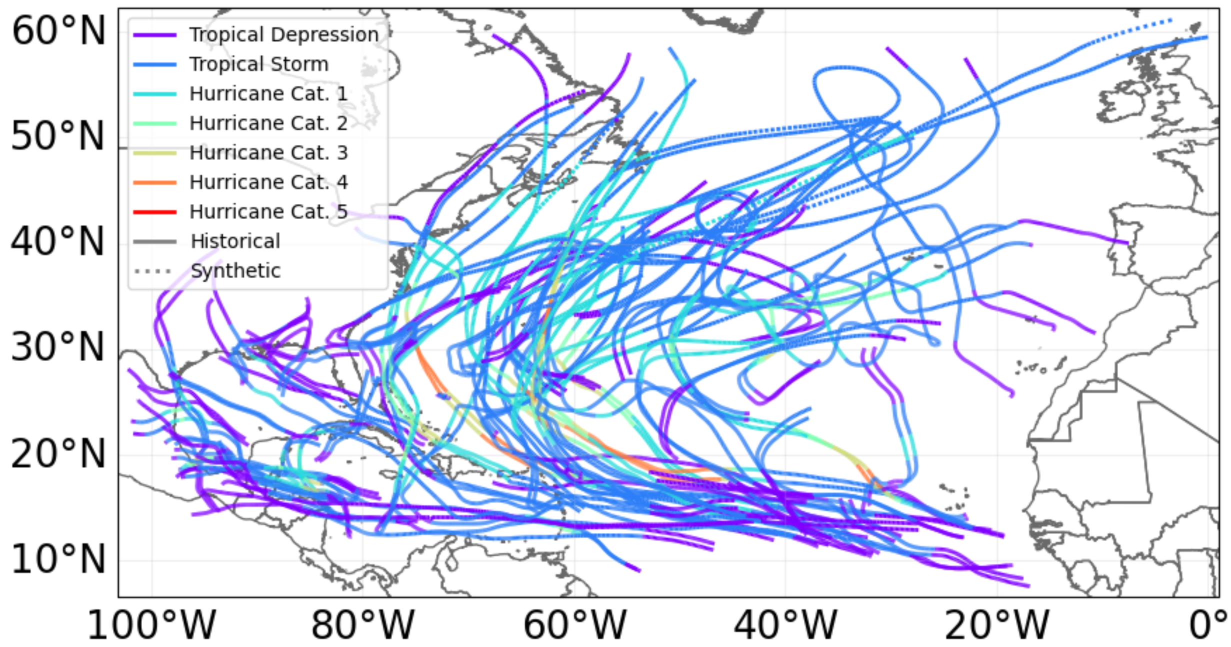
Define the hazard

Let us define a tropical cyclone hazard using the TropCyclone and TCTracks modules.

```
from climada.hazard import TCTracks, TropCyclone, Centroids

# Load histrocial tropical cyclone tracks from ibtracs over the North Atlantic basin␣
↪between 2010-2012
ibtracks_na = TCTracks.from_ibtracs_netcdf(provider='usa', basin='NA', year_
↪range=(2010, 2012), correct_pres=True)
print('num tracks hist:', ibtracks_na.size)

ibtracks_na.equal_timestep(0.5)  # Interpolation to make the track smooth and to␣
↪allow applying calc_perturbed_trajectories
# Add randomly generated tracks using the calc_perturbed_trajectories method (1 per␣
↪historical track)
ibtracks_na.calc_perturbed_trajectories(nb_synth_tracks=1)
print('num tracks hist+syn:', ibtracks_na.size)
```

```
2023-01-26 11:57:25,332 - climada.hazard.tc_tracks - WARNING - `correct_pres` is␣
↪deprecated. Use `estimate_missing` instead.
2023-01-26 11:57:26,701 - climada.hazard.tc_tracks - INFO - Progress: 10%
2023-01-26 11:57:26,905 - climada.hazard.tc_tracks - INFO - Progress: 20%
2023-01-26 11:57:27,075 - climada.hazard.tc_tracks - INFO - Progress: 30%
2023-01-26 11:57:27,229 - climada.hazard.tc_tracks - INFO - Progress: 40%
2023-01-26 11:57:27,390 - climada.hazard.tc_tracks - INFO - Progress: 50%
2023-01-26 11:57:27,542 - climada.hazard.tc_tracks - INFO - Progress: 60%
2023-01-26 11:57:27,713 - climada.hazard.tc_tracks - INFO - Progress: 70%
2023-01-26 11:57:27,876 - climada.hazard.tc_tracks - INFO - Progress: 80%
2023-01-26 11:57:28,032 - climada.hazard.tc_tracks - INFO - Progress: 90%
2023-01-26 11:57:28,197 - climada.hazard.tc_tracks - INFO - Progress: 100%
num tracks hist: 60
2023-01-26 11:57:28,229 - climada.hazard.tc_tracks - INFO - Interpolating 60 tracks␣
↪to 0.5h time steps.
2023-01-26 11:57:32,223 - climada.hazard.tc_tracks_synth - INFO - Computing 60␣
↪synthetic tracks.
num tracks hist+syn: 120
```

```
# not needed for calculations
# visualize tracks
ax = ibtracks_na.plot()
ax.get_legend()._loc = 2
```



From the tracks, we generate the hazards (the tracks are only the coordinates of the center of the cyclones, the full cyclones however affects a region around the tracks).

First thing we define the set of centroids which are geographical points where the hazard has a defined value. In our case, we want to define windspeeds from the tracks.

Remember: In the impact computations, for each exposure geographical point, one must assign a centroid from the hazard. By default, each exposure is assigned to the closest centroid from the hazard. But one can also define manually which centroid is assigned to which exposure point.

Examples:

- Define the exposures from a given source (e.g., raster of asset values from LitPop). Define the hazard centroids from the exposures' geolocations (e.g. compute Tropical Cyclone windspeed at each raster point and assign centroid to each raster point).
- Define the exposures from a given source (e.g. houses position and value). Define the hazard from a given source (e.g. where lanslides occur). Use a metric to assign to each exposures point a hazard centroid (all houses in a radius of 5km around the lanslide are assigned to this centroid, if a house is within 5km of two landslides, choose the closest one).
- Define a geographical raster. Define the exposures value on this raster. Define the hazard centroids on the geographical raster.

We shall pursue with the first case (Litpop + TropicalCyclone)

Hint: computing the wind speeds in many locations for many tc tracks is a computationally costly operation. Thus, we should define centroids only where we also have an exposure.

```
# Define the centroids from the exposures position
lat = exp_lp.gdf['latitude'].values
lon = exp_lp.gdf['longitude'].values
centrs = Centroids.from_lat_lon(lat, lon)
centrs.check()
```

```
# Using the tracks, compute the windspeed at the location of the centroids
tc = TropCyclone.from_tracks(ibtracks_na, centroids=centrs)
tc.check()
```

```
2023-01-26 11:58:43,439 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2023-01-26 11:58:44,363 - climada.util.coordinates - INFO - dist_to_coast: UTM 32616␣
↪(1/3)
2023-01-26 11:58:44,606 - climada.util.coordinates - INFO - dist_to_coast: UTM 32617␣
↪(2/3)
2023-01-26 11:58:45,335 - climada.util.coordinates - INFO - dist_to_coast: UTM 32618␣
↪(3/3)
2023-01-26 11:58:45,767 - climada.hazard.trop_cyclone - INFO - Mapping 120 tracks to␣
↪1388 coastal centroids.
2023-01-26 11:58:46,030 - climada.hazard.trop_cyclone - INFO - Progress: 10%
2023-01-26 11:58:46,203 - climada.hazard.trop_cyclone - INFO - Progress: 20%
2023-01-26 11:58:46,499 - climada.hazard.trop_cyclone - INFO - Progress: 30%
2023-01-26 11:58:46,784 - climada.hazard.trop_cyclone - INFO - Progress: 40%
2023-01-26 11:58:47,231 - climada.hazard.trop_cyclone - INFO - Progress: 50%
2023-01-26 11:58:47,353 - climada.hazard.trop_cyclone - INFO - Progress: 60%
2023-01-26 11:58:47,546 - climada.hazard.trop_cyclone - INFO - Progress: 70%
2023-01-26 11:58:47,667 - climada.hazard.trop_cyclone - INFO - Progress: 80%
2023-01-26 11:58:48,102 - climada.hazard.trop_cyclone - INFO - Progress: 90%
2023-01-26 11:58:48,418 - climada.hazard.trop_cyclone - INFO - Progress: 100%
```

Hint: The operation of computing the windspeed in different location is in general computationally expensive. Hence, if you have a lot of tropical cyclone tracks, you should first make sure that all your tropical cyclones actually affect your exposure (remove those that don't). Then, be careful when defining the centroids. For a large country like China, there is no need for centroids 500km inland (no tropical cyclones gets so far).

Impact function

For Tropical Cyclones, some calibrated default impact functions exist. Here we will use the one from Emanuel (2011).

```
from climada.entity import ImpactFuncSet, ImpfTropCyclone
# impact function TC
impf_tc = ImpfTropCyclone.from_emanuel_usa()

# add the impact function to an Impact function set
impf_set = ImpactFuncSet([impf_tc])
impf_set.check()
```

Recall that the exposures, hazards and impact functions must be matched in the impact calculations. Here it is simple, since there is a single impact function for all the hazards. We must simply make sure that the exposure is assigned this impact function through renaming the `impf\_` column from the hazard type of the impact function in the impact function set and set the values of the column to the id of the impact function.

```
# Get the hazard type and hazard id
[haz_type] = impf_set.get_hazard_types()
[haz_id] = impf_set.get_ids()[haz_type]
print(f"hazard type: {haz_type}, hazard id: {haz_id}")
```

```
hazard type: TC, hazard id: 1
```

```
# Exposures: rename column and assign id
exp_lp.gdf.rename(columns={"impf_": "impf_" + haz_type}, inplace=True)
exp_lp.gdf['impf_' + haz_type] = haz_id
exp_lp.check()
exp_lp.gdf.head()
```

```
2023-01-26 11:58:48,792 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:58:48,792 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:58:48,792 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:58:48,792 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
          value                       geometry   latitude  longitude  region_id  \
0  1.077368e+05  POINT (-81.37500 21.70833)  21.708333 -81.375000        192
1  1.671873e+06  POINT (-81.54167 21.62500)  21.625000 -81.541667        192
2  3.421208e+06  POINT (-82.95833 21.87500)  21.875000 -82.958333        192
3  1.546590e+07  POINT (-82.87500 21.87500)  21.875000 -82.875000        192
4  7.168305e+07  POINT (-82.79167 21.87500)  21.875000 -82.791667        192

   impf_TC
0        1
1        1
2        1
3        1
4        1
```

Impact computation

We are finally ready for the impact computation. This is the simplest step. Just give the exposure, impact function and hazard to the ImpactCalc.impact() method.

Note: we did not specifically assign centroids to the exposures. Hence, the default is used - each exposure is associated with the closest centroids. Since we defined the centroids from the exposures, this is a one-to-one mapping.

Note: we did not define an `Entity` in this impact calculations. Recall that `Entity` is a container class for ***Exposures***, ***Impact Functions***, ***Discount Rates*** and ***Measures***. Since we had only one Exposure and one Impact Function, the container would not have added any value, but for more complex projects, the Entity class is very useful.

```
# Compute impact
from climada.engine import ImpactCalc
imp = ImpactCalc(exp_lp, impf_set, tc).impact(save_mat=False)  # Do not save the␣
↪results geographically resolved (only aggregate values)
```

```
2023-01-26 11:58:48,877 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 1388 centroids.
2023-01-26 11:58:48,877 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪4164 assets (>0) and 120 events.
```

```
exp_lp.gdf
```

```
             value                       geometry   latitude  longitude  \
0     1.077368e+05  POINT (-81.37500 21.70833)  21.708333 -81.375000
1     1.671873e+06  POINT (-81.54167 21.62500)  21.625000 -81.541667
2     3.421208e+06  POINT (-82.95833 21.87500)  21.875000 -82.958333
3     1.546590e+07  POINT (-82.87500 21.87500)  21.875000 -82.875000
```

(continues on next page)

(continued from previous page)

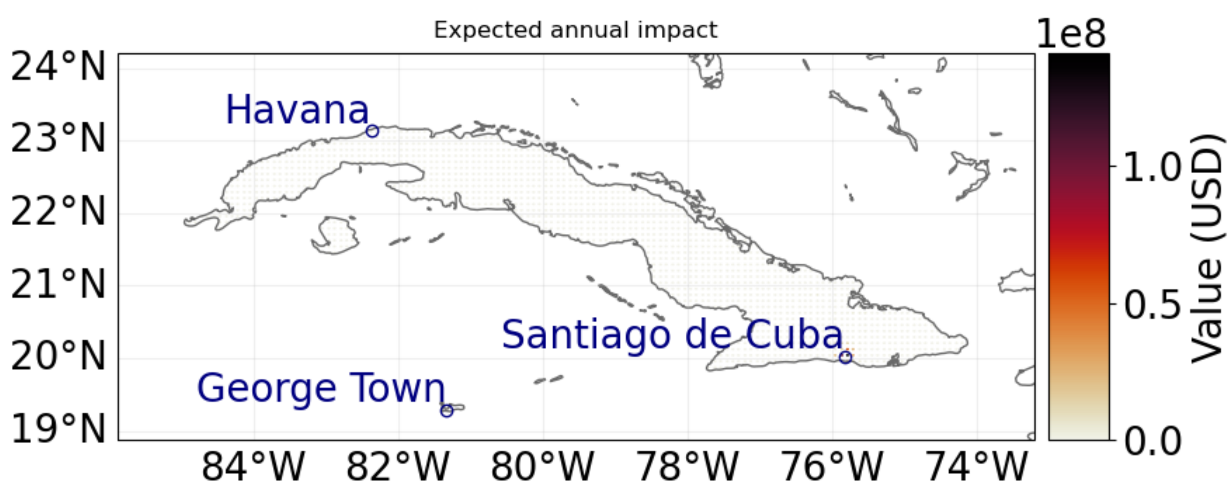
```
4     7.168305e+07  POINT (-82.79167 21.87500)  21.875000 -82.791667
...            ...                         ...        ...        ...
1383  1.496797e+06  POINT (-78.62500 22.54167)  22.541667 -78.625000
1384  5.387835e+07  POINT (-78.37500 22.54167)  22.541667 -78.375000
1385  4.077093e+06  POINT (-78.45833 22.45833)  22.458333 -78.458333
1386  2.249377e+07  POINT (-78.37500 22.45833)  22.458333 -78.375000
1387  6.191982e+06  POINT (-78.29167 22.45833)  22.458333 -78.291667

      region_id  impf_TC  centr_TC
0           192        1         0
1           192        1         1
2           192        1         2
3           192        1         3
4           192        1         4
...         ...      ...       ...
1383        192        1      1383
1384        192        1      1384
1385        192        1      1385
1386        192        1      1386
1387        192        1      1387

[1388 rows x 7 columns]
```

For example we can now obtain the aggregated average annual impact or plot the average annual impact in each exposure location.

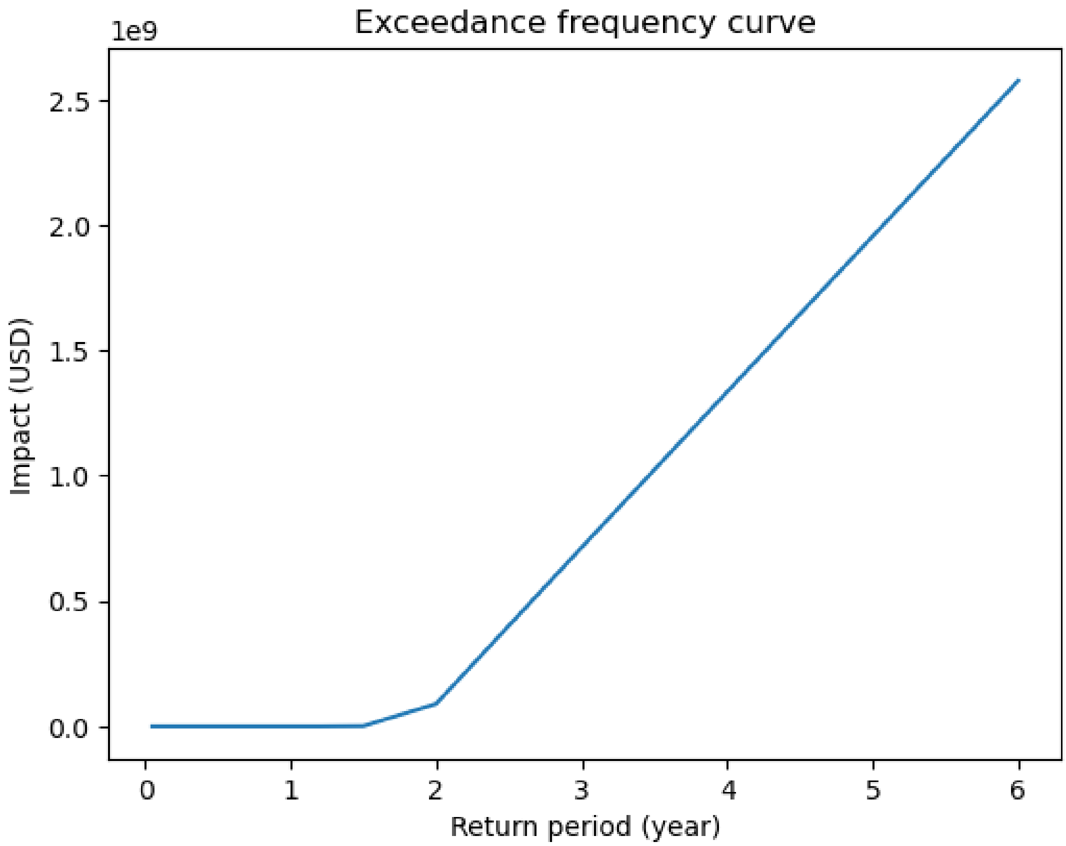
```
print(f"Aggregated average annual impact: {round(imp.aai_agg,0)} $")
```

```
Aggregated average annual impact: 563366225.0 $
```

```
imp.plot_hexbin_eai_exposure(buffer=1);
```




```
# Compute exceedance frequency curve
freq_curve = imp.calc_freq_curve()
freq_curve.plot();
```



Impact concatenation

There can be cases in which an impact function for a given hazard type is not constant throughout the year. This is for example the case in the context of agriculture. For example, if a crop is already harvested the impact of a certain weather event can be much lower or even zero. For such situations of two or more different impact functions for the same hazard and exposure type, it can be useful to split the events into subsets and compute impacts separately. In order to then analyze the total impact, the different impact subsets can be concatenated using the `Impact.concat` method. This is done here for the hypothetical example using LitPop as exposure and TCs as hazard. For illustration purposes, we misuse the LitPop exposure in this case as exposure of a certain crop. We assume a constant harvest day (17 October) after which the impact function is reduced by a factor of 10.

First, we prepare the hazard subsets.

```
from datetime import datetime, date
import pandas as pd

#set a harvest date
harvest_DOY=290 #17 October

#loop over all events an check if they happened before or after harvest
event_ids_post_harvest=[]
event_ids_pre_harvest=[]
for event_id in tc.event_id:
            event_date = tc.date[np.where(tc.event_id==event_id)[0][0]]
            day_of_year = event_date - date(datetime.fromordinal(event_date).year, 1,␣
↪1).toordinal() + 1
```

(continues on next page)

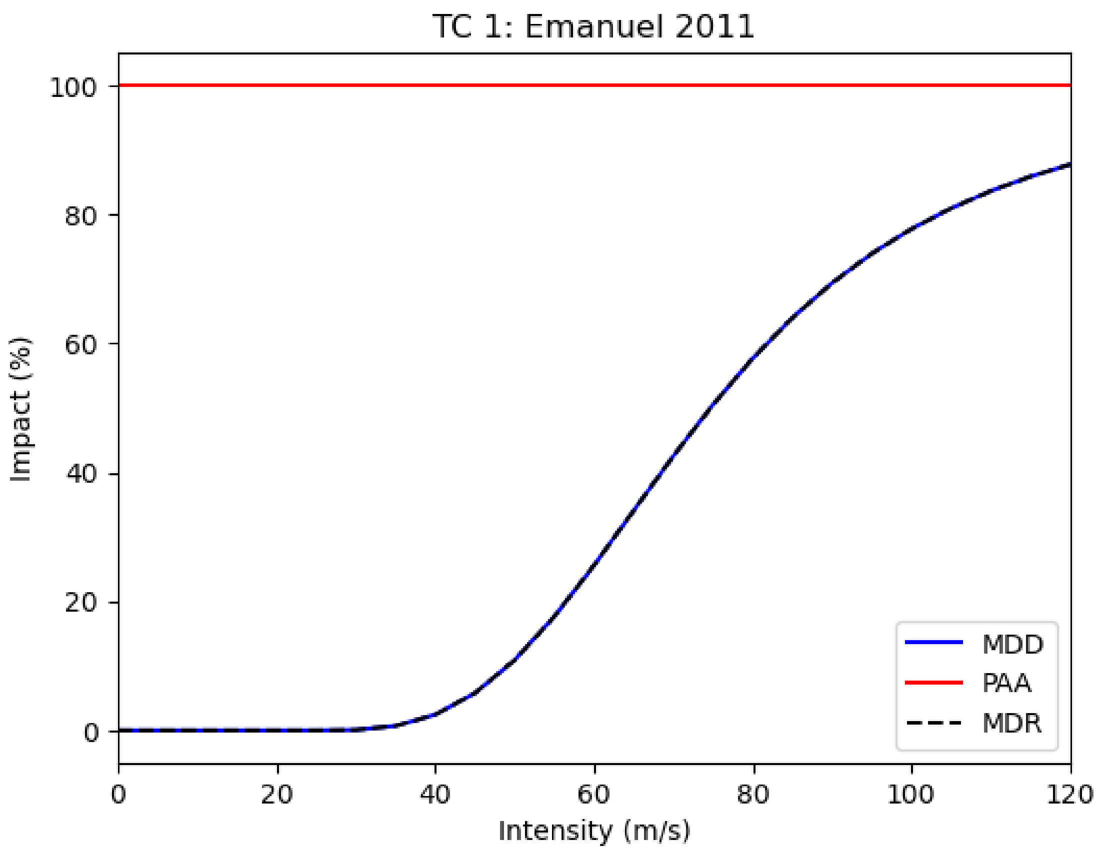
(continued from previous page)

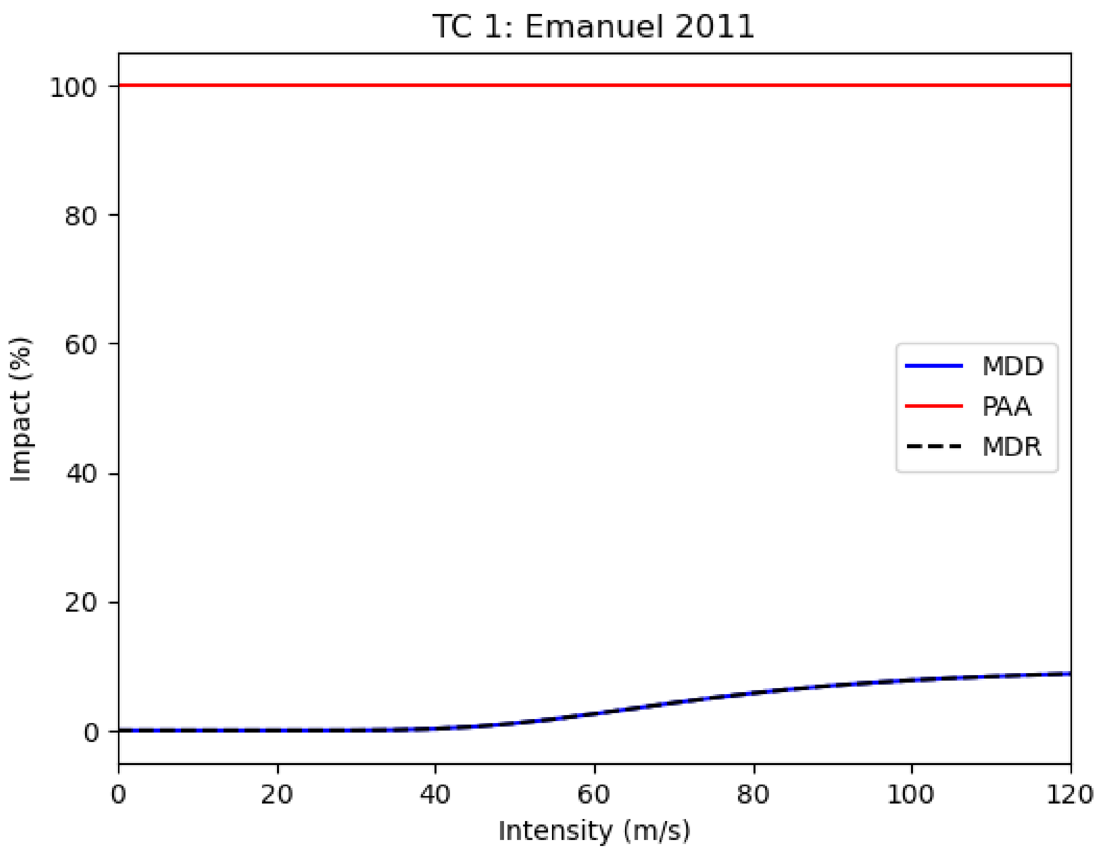
```
            if day_of_year > harvest_DOY:
                event_ids_post_harvest.append(event_id)
            else:
                event_ids_pre_harvest.append(event_id)

tc_post_harvest=tc.select(event_id=event_ids_post_harvest)
tc_pre_harvest=tc.select(event_id=event_ids_pre_harvest)
#print('pre-harvest:', tc_pre_harvest.event_name)
#print('post-harvest:', tc_post_harvest.event_name)
```

Now we get two different impact functions, one valid for the exposed crop before harvest and one after harvest. Then, we compute the impacts for both phases separately.

```
from climada.engine import Impact
# impact function TC
impf_tc = ImpfTropCyclone.from_emanuel_usa()
# impact function TC after harvest is by factor 0.5 smaller
impf_tc_posth = ImpfTropCyclone.from_emanuel_usa()
impf_tc_posth.mdd = impf_tc.mdd*0.1
# add the impact function to an Impact function set
impf_set = ImpactFuncSet([impf_tc])
impf_set_posth = ImpactFuncSet([impf_tc_posth])
impf_set.check()
impf_set_posth.check()

#plot
impf_set.plot()
impf_set_posth.plot()

# Compute impacts
imp_preh = ImpactCalc(exp_lp, impf_set, tc_pre_harvest).impact(save_mat=True)
imp_posth = ImpactCalc(exp_lp, impf_set_posth, tc_post_harvest).impact(save_mat=True)
```

```
2023-01-26 11:58:52,364 - climada.entity.exposures.base - INFO - Exposures matching␣
↪centroids already found for TC
2023-01-26 11:58:52,364 - climada.entity.exposures.base - INFO - Existing centroids␣
↪will be overwritten for TC
2023-01-26 11:58:52,364 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 1388 centroids.
2023-01-26 11:58:52,370 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪4164 assets (>0) and 106 events.
2023-01-26 11:58:52,379 - climada.entity.exposures.base - INFO - Exposures matching␣
↪centroids already found for TC
2023-01-26 11:58:52,379 - climada.entity.exposures.base - INFO - Existing centroids␣
↪will be overwritten for TC
2023-01-26 11:58:52,382 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 1388 centroids.
2023-01-26 11:58:52,388 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪4164 assets (>0) and 14 events.
```

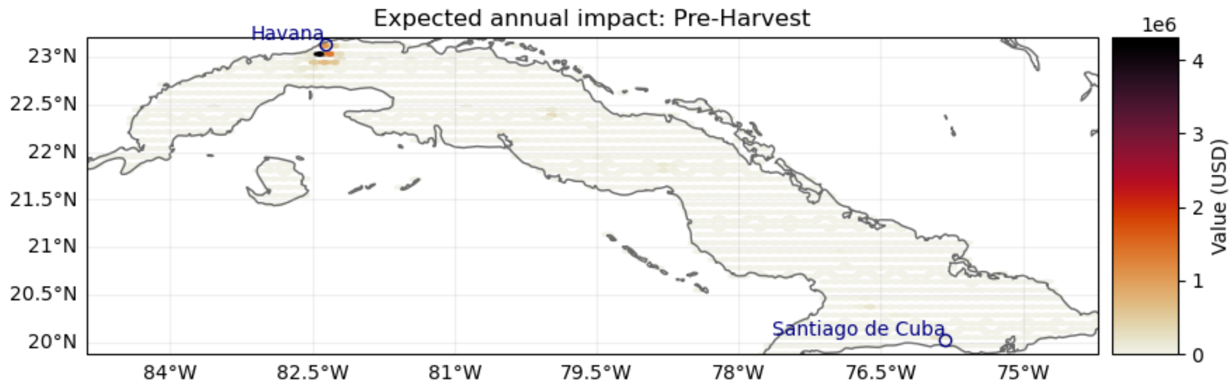

TC 1: Emanuel 2011
Impact (%)
Intensity (m/s)
0
20
40
60
80
100
120
MDD
PAA
MDR

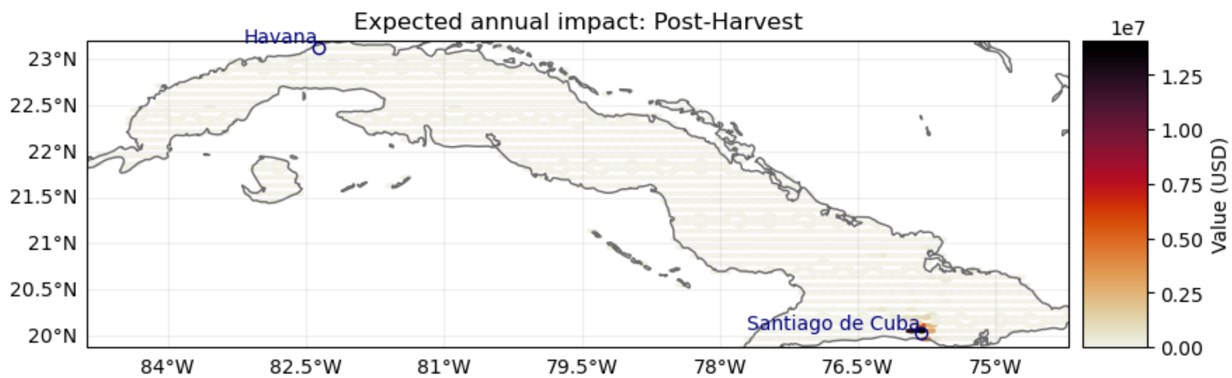


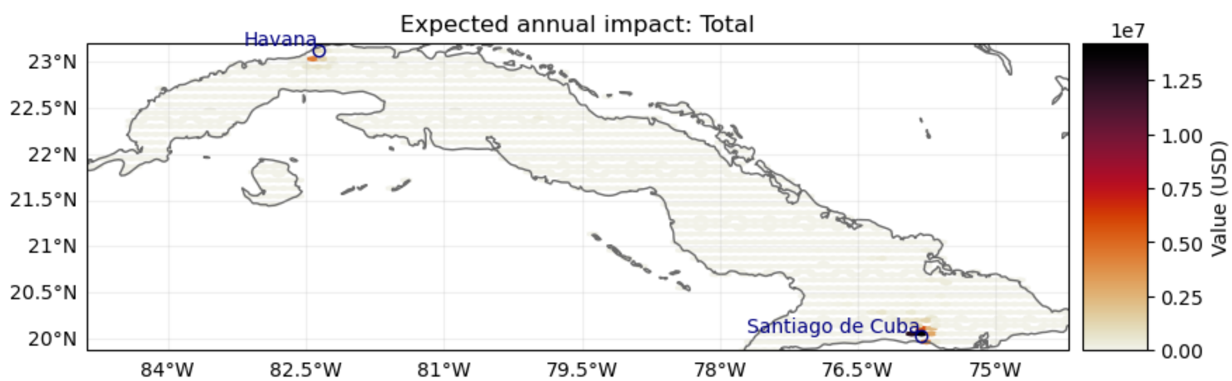
Now, we can concatenate the impacts again and plot the results

```
# Concatenate impacts again
imp_tot = Impact.concat([imp_preh,imp_posth])

#plot result
import matplotlib.pyplot as plt
ax=imp_preh.plot_hexbin_eai_exposure(gridsize=100,adapt_fontsize=False)
ax.set_title('Expected annual impact: Pre-Harvest')
ax=imp_posth.plot_hexbin_eai_exposure(gridsize=100,adapt_fontsize=False)
ax.set_title('Expected annual impact: Post-Harvest')
ax=imp_tot.plot_hexbin_eai_exposure(gridsize=100,adapt_fontsize=False)
ax.set_title('Expected annual impact: Total')
```

```
Text(0.5, 1.0, 'Expected annual impact: Total')
```







10.1.7 Quick examples - points, raster, custom

User defined point exposure and Tropical Cyclone hazard

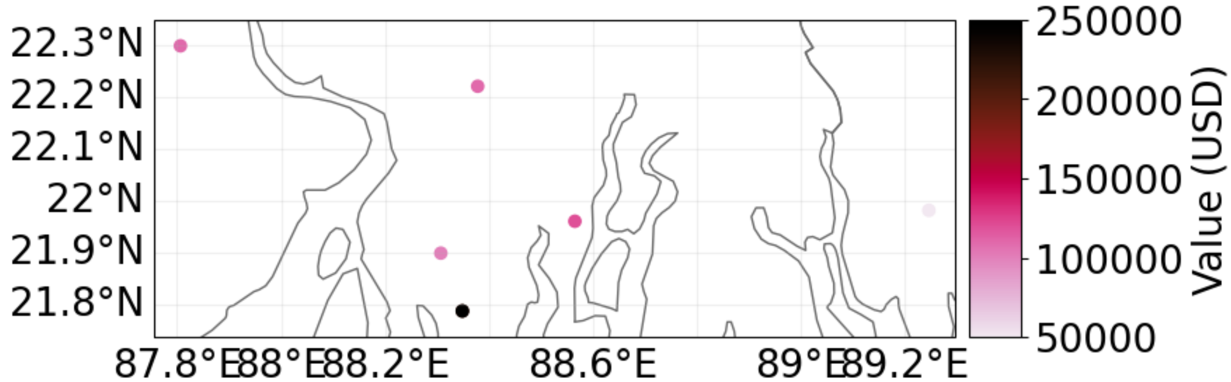
```
%matplotlib inline
# EXAMPLE: POINT EXPOSURES WITH POINT HAZARD
import numpy as np
from climada.entity import Exposures, ImpactFuncSet, IFTropCyclone
from climada.hazard import Centroids, TCTracks, TropCyclone
from climada.engine import ImpactCalc

# Set Exposures in points
exp_pnt = Exposures(crs='epsg:4326') #set coordinate system
exp_pnt.gdf['latitude'] = np.array([21.899326, 21.960728, 22.220574, 22.298390, 21.
↪787977, 21.787977, 21.981732])
exp_pnt.gdf['longitude'] = np.array([88.307422, 88.565362, 88.378337, 87.806356, 88.
↪348835, 88.348835, 89.246521])
exp_pnt.gdf['value'] = np.array([1.0e5, 1.2e5, 1.1e5, 1.1e5, 2.0e5, 2.5e5, 0.5e5])
exp_pnt.check()
exp_pnt.plot_scatter(buffer=0.05)

# Set Hazard in Exposures points
# set centroids from exposures coordinates
centr_pnt = Centroids.from_lat_lon(exp_pnt.gdf.latitude.values, exp_pnt.gdf.longitude.
↪values, exp_pnt.crs)
# compute Hazard in that centroids
tr_pnt = TCTracks.from_ibtracs_netcdf(storm_id='2007314N10093')
tc_pnt = TropCyclone.from_tracks(tr_pnt, centroids=centr_pnt)
tc_pnt.check()
ax_pnt = tc_pnt.centroids.plot(c=np.array(tc_pnt.intensity[0,:].todense()).squeeze())
↪# plot intensity per point
ax_pnt.get_figure().colorbar(ax_pnt.collections[0], fraction=0.0175, pad=0.02).set_
↪label('Intensity (m/s)') # add colorbar

# Set impact function
impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_pnt = ImpactFuncSet([impf_tc])
impf_pnt.check()

# Get the hazard type and hazard id
[haz_type] = impf_set.get_hazard_types()
[haz_id] = impf_set.get_ids()[haz_type]
# Exposures: rename column and assign id
exp_lp.gdf.rename(columns={"impf_": "impf_" + haz_type}, inplace=True)
exp_lp.gdf['impf_' + haz_type] = haz_id
exp_lp.gdf.head()

# Compute Impact
imp_pnt = ImpactCalc(exp_pnt, impf_pnt, tc_pnt).impact()
# nearest neighbor of exposures to centroids gives identity
print('Nearest neighbor hazard.centroids indexes for each exposure:', exp_pnt.gdf.
↪centr_TC.values)
imp_pnt.plot_scatter_eai_exposure(ignore_zero=False, buffer=0.05);
```

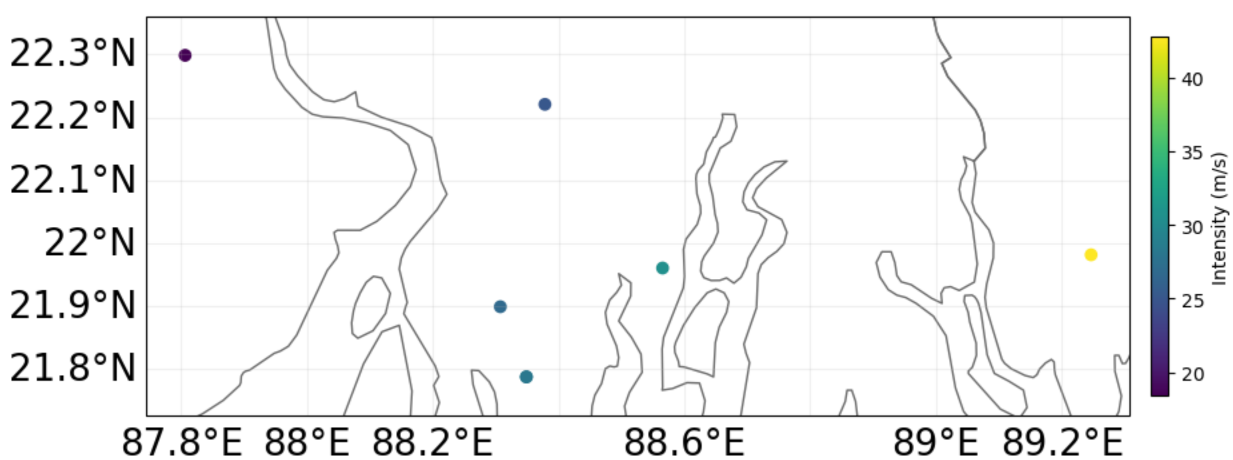
```
2023-01-26 11:59:01,455 - climada.entity.exposures.base - INFO - Setting impf_ to␣
↪default impact functions ids 1.
2023-01-26 11:59:01,457 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:59:01,458 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:59:01,460 - climada.entity.exposures.base - INFO - deductible not set.
```

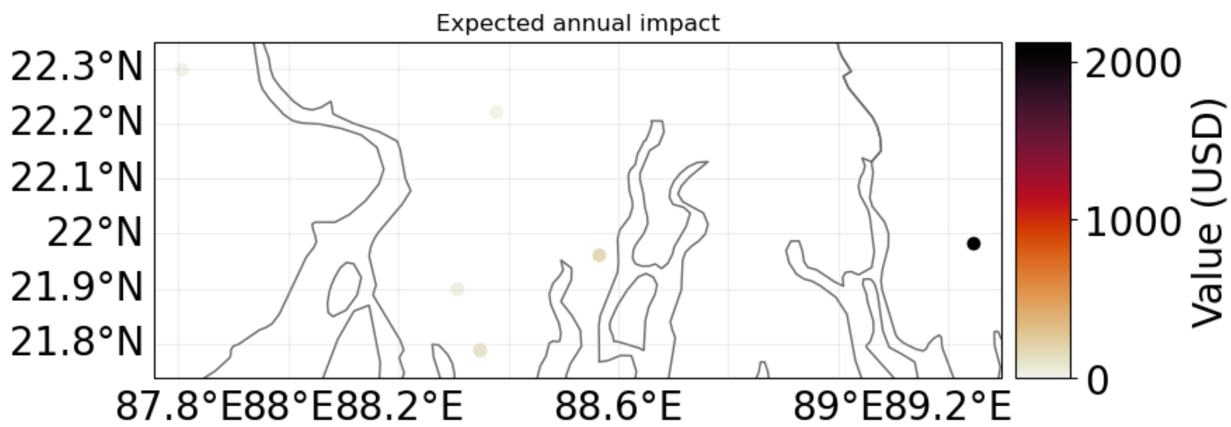
(continues on next page)

(continued from previous page)

```
2023-01-26 11:59:01,463 - climada.entity.exposures.base - INFO - geometry not set.
2023-01-26 11:59:01,464 - climada.entity.exposures.base - INFO - region_id not set.
2023-01-26 11:59:01,466 - climada.entity.exposures.base - INFO - centr_ not set.
2023-01-26 11:59:03,801 - climada.hazard.tc_tracks - INFO - Progress: 100%
2023-01-26 11:59:03,846 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2023-01-26 11:59:04,466 - climada.util.coordinates - INFO - dist_to_coast: UTM 32645␣
↪(1/1)
2023-01-26 11:59:04,580 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 7␣
↪coastal centroids.
2023-01-26 11:59:04,595 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 11:59:05,457 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 11:59:05,458 - climada.entity.exposures.base - INFO - Matching 7 exposures␣
↪with 7 centroids.
2023-01-26 11:59:05,463 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪21 assets (>0) and 1 events.
Nearest neighbor hazard.centroids indexes for each exposure: [0 1 2 3 4 5 6]
```







Raster from file

```
# EXAMPLE: RASTER EXPOSURES WITH RASTER HAZARD
from rasterio.warp import Resampling
from climada.entity import LitPop, ImpactFuncSet, ImpactFunc
from climada.hazard import Hazard
from climada.engine import Impact
from climada.util.constants import HAZ_DEMO_FL

# Exposures belonging to a raster (the raser information is contained in the meta␣
↪attribute)
exp_ras = LitPop.from_countries(countries=['VEN'], res_arcsec=300, fin_mode='income_
↪group')
exp_ras.gdf.reset_index()
exp_ras.check()
exp_ras.plot_raster()
print('\n Raster properties exposures:', exp_ras.meta)

# Initialize hazard object with haz_type = 'FL' (for Flood)
hazard_type='FL'
# Load a previously generated (either with CLIMADA or other means) hazard
# from file (HAZ_DEMO_FL) and resample the hazard raster to the exposures' ones
# Hint: check how other resampling methods affect to final impact
haz_ras = Hazard.from_raster([HAZ_DEMO_FL], haz_type=hazard_type, dst_crs=exp_ras.
↪meta['crs'], transform=exp_ras.meta['transform'],
                             width=exp_ras.meta['width'], height=exp_ras.meta['height
↪'],
                             resampling=Resampling.nearest)
haz_ras.intensity[haz_ras.intensity==-9999] = 0 # correct no data values
haz_ras.check()
haz_ras.plot_intensity(1)
print('Raster properties centroids:', haz_ras.centroids.meta)

# Set dummy impact function
intensity = np.linspace(0, 10, 100)
mdd = np.linspace(0, 10, 100)
paa = np.ones(intensity.size)
impf_dum = ImpactFunc(hazard_type, haz_id, intensity, mdd, paa, "m", "dummy")
# Add the impact function to the impact function set
impf_ras = ImpactFuncSet([impf_dum])
```

(continues on next page)

(continued from previous page)

```
impf_ras.check()

# Exposures: rename column and assign id
exp_lp.gdf.rename(columns={"impf_": "impf_" + hazard_type}, inplace=True)
exp_lp.gdf['impf_' + haz_type] = haz_id
exp_lp.gdf.head()

# Compute impact
imp_ras = ImpactCalc(exp_ras, impf_ras, haz_ras).impact(save_mat=False)
# nearest neighbor of exposures to centroids is not identity because litpop does not␣
↪contain data outside the country polygon
print('\n Nearest neighbor hazard.centroids indexes for each exposure:', exp_ras.gdf.
↪centr_FL.values)
imp_ras.plot_raster_eai_exposure();
```

```
2023-01-26 11:59:11,285 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: VEN (862)...

2023-01-26 11:59:13,749 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:13,751 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:13,783 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:13,785 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:13,787 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:13,850 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:13,852 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:13,926 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:13,928 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:13,984 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:13,985 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,045 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,046 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,105 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,108 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,167 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,168 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,220 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,221 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2023-01-26 11:59:14,252 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:14,253 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,255 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,283 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:14,284 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,286 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,314 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:14,315 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,317 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,348 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:14,349 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,351 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,381 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:14,383 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,386 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,416 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:14,417 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,419 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,491 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,493 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,556 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,557 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,622 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,624 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,703 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,705 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,779 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,780 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,854 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
```

(continues on next page)

(continued from previous page)

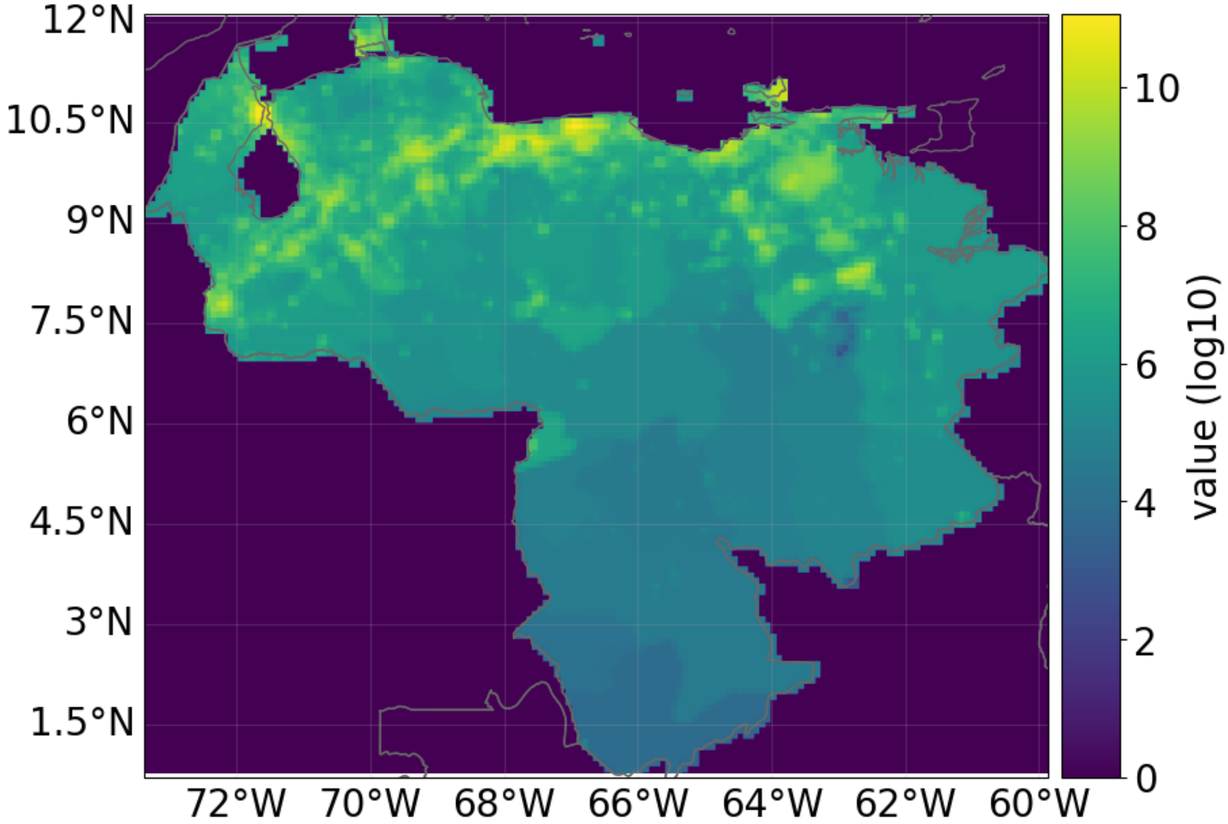
```
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,855 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,922 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,924 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,988 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,990 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:15,026 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:15,028 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:15,029 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:15,065 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:15,066 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:15,069 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:15,130 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:15,131 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:15,200 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:15,202 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:15,263 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
```

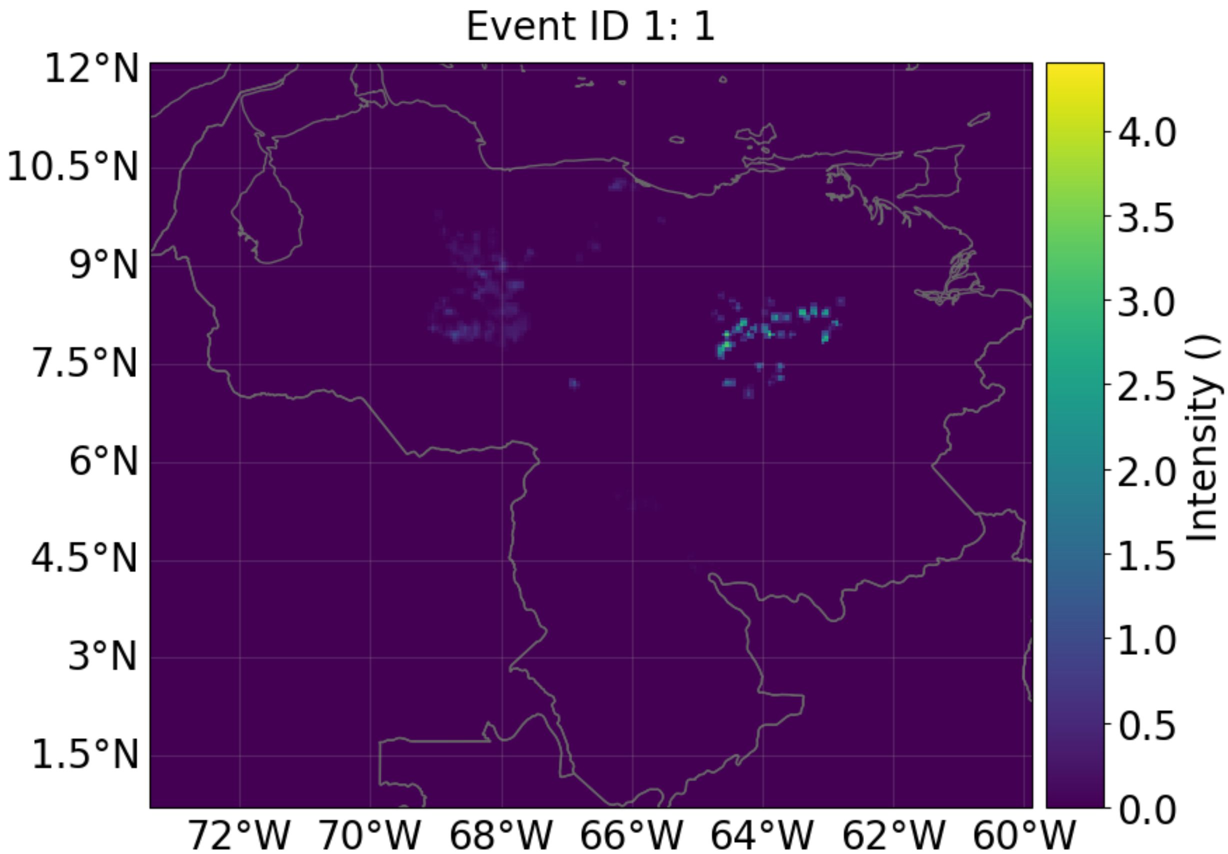
```
2023-01-26 11:59:15,265 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:15,289 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:15,833 - climada.util.finance - INFO - GDP VEN 2014: 4.824e+11.
2023-01-26 11:59:15,909 - climada.util.finance - INFO - Income group VEN 2018: 3.
2023-01-26 11:59:15,933 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-01-26 11:59:15,934 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:59:15,936 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:59:15,937 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:59:15,939 - climada.entity.exposures.base - INFO - centr_ not set.
2023-01-26 11:59:15,949 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-01-26 11:59:15,951 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:59:15,952 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:59:15,953 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:59:15,955 - climada.entity.exposures.base - INFO - centr_ not set.
2023-01-26 11:59:15,962 - climada.util.coordinates - INFO - Raster from resolution 0.
↪08333332999999987 to 0.08333332999999987.

 Raster properties exposures: {'width': 163, 'height': 138, 'crs': <Geographic 2D␣
```

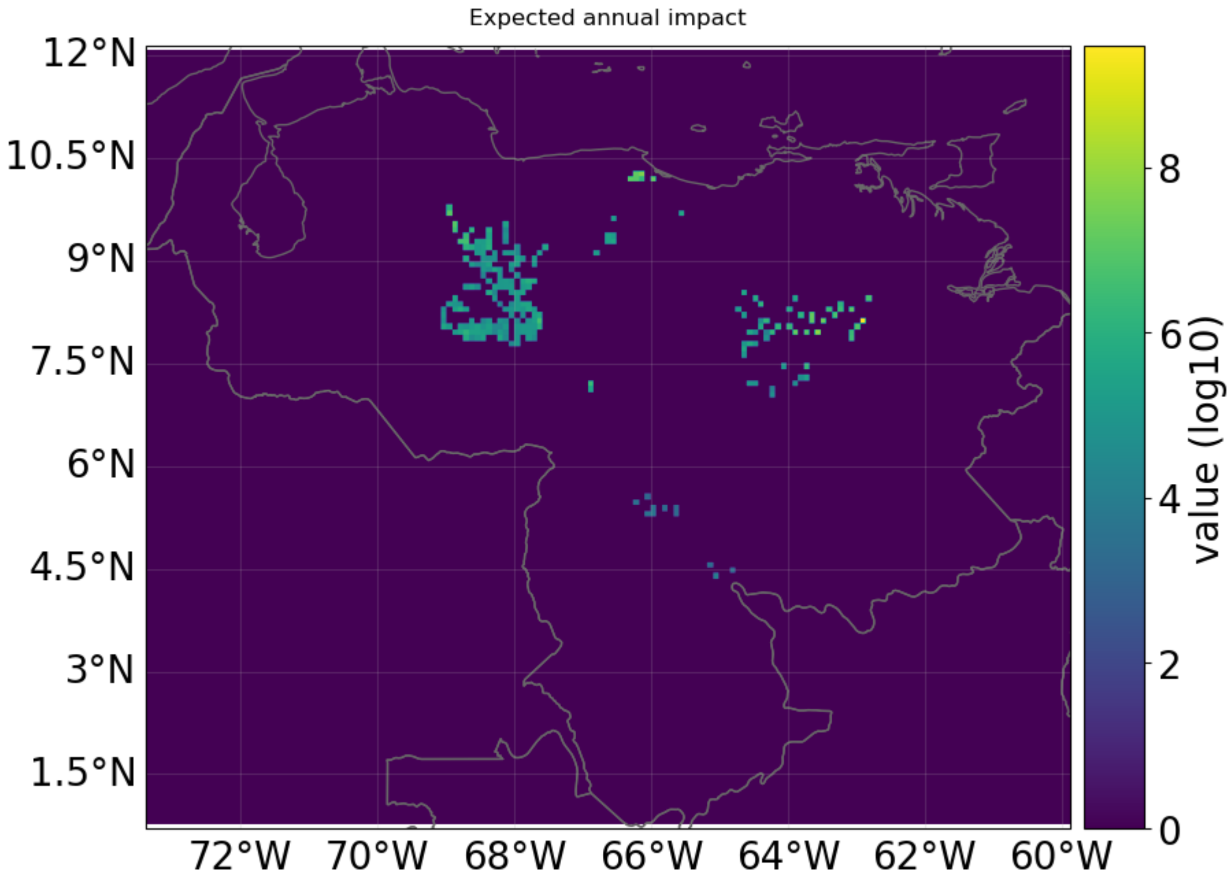
(continues on next page)

(continued from previous page)

```
↪CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- undefined
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
, 'transform': Affine(0.08333333000000209, 0.0, -73.41666666500001,
       0.0, -0.08333332999999987, 12.166666665)}
2023-01-26 11:59:23,374 - climada.util.coordinates - INFO - Reading C:\Users\
↪F80840370\climada\demo\data\SC22000_VE__M1.grd.gz
2023-01-26 11:59:25,519 - climada.util.coordinates - INFO - Reading C:\Users\
↪F80840370\climada\demo\data\SC22000_VE__M1.grd.gz
Raster properties centroids: {'driver': 'GSBG', 'dtype': 'float32', 'nodata': 1.
↪701410009187828e+38, 'width': 163, 'height': 138, 'count': 1, 'crs': <Geographic 2D␣
↪CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- undefined
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
, 'transform': Affine(0.08333333000000209, 0.0, -73.41666666500001,
       0.0, -0.08333332999999987, 12.166666665)}
2023-01-26 11:59:28,695 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard FL. Using the anonymous 'impf_' column.
2023-01-26 11:59:28,695 - climada.entity.exposures.base - INFO - Matching 10772␣
↪exposures with 22494 centroids.
2023-01-26 11:59:28,704 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪32310 assets (>0) and 1 events.

 Nearest neighbor hazard.centroids indexes for each exposure: [  39   40   41 ...␣
↪3387 3551 2721]
2023-01-26 11:59:28,714 - climada.util.coordinates - INFO - Raster from resolution 0.
↪08333332999999987 to 0.08333332999999987.
```


12°N
10.5°N
9°N
7.5°N
6°N
4.5°N
3°N
1.5°N
72°W
70°W
68°W
66°W
64°W
62°W
60°W
value (log10)
0
2
4
6
8
10


Event ID 1: 1
12°N
10.5°N
9°N
7.5°N
6°N
4.5°N
3°N
1.5°N
72°W
70°W
68°W
66°W
64°W
62°W
60°W
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Intensity ()



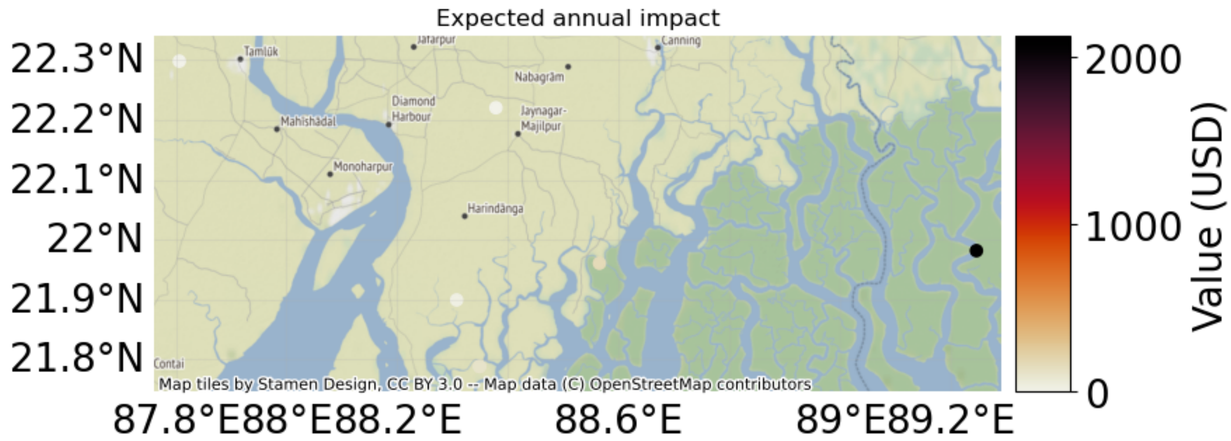
10.1.8 VISUALIZATION

Making plots

The expected annual impact per exposure can be visualized through different methods: `plot_hexbin_eai_exposure()`, `plot_scatter_eai_exposur()`, `plot_raster_eai_exposure()` and `plot_basemap_eai_exposure()` (similarly as with `Exposures`).

```
imp_pnt.plot_basemap_eai_exposure(buffer=5000);
```

```
2023-01-26 11:59:40,355 - climada.util.coordinates - INFO - Setting geometry points.
2023-01-26 11:59:40,364 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2023-01-26 11:59:43,294 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```



Making videos

Given a fixed exposure and impact functions, a sequence of hazards can be visualized hitting the exposures.

```
# exposure
from climada.entity import add_sea
from climada_petals.entity import BlackMarble

exp_video = BlackMarble()
exp_video.set_countries(['Cuba'], 2016, res_km=2.5)
exp_video.check()

# impact function
impf_def = ImpfTropCyclone.from_emanuel_usa()
impfs_video = ImpactFuncSet([impf_def])
impfs_video.check()

# compute sequence of hazards using TropCyclone video_intensity method
exp_sea = add_sea(exp_video, (100, 5))
centr_video = Centroids.from_lat_lon(exp_sea.gdf.latitude.values, exp_sea.gdf.
↪longitude.values)
centr_video.check()

track_name = '2017242N16333'
tr_irma = TCTracks.from_ibtracs_netcdf(provider='usa', storm_id=track_name) # IRMA␣
↪2017

tc_video = TropCyclone()
tc_list, _ = tc_video.video_intensity(track_name, tr_irma, centr_video) # empty file␣
↪name to not to write the video

# generate video of impacts
file_name='./results/irma_imp_fl.gif'
imp_video = Impact()
imp_list = imp_video.video_direct_impact(exp_video, impfs_video, tc_list, file_name)
```

```
2023-01-26 11:59:44,414 - climada.util.finance - INFO - GDP CUB 2016: 9.137e+10.
2023-01-26 11:59:44,483 - climada.util.finance - INFO - Income group CUB 2016: 3.
2023-01-26 11:59:44,483 - climada_petals.entity.exposures.black_marble - INFO -␣
↪Nightlights from NASA's earth observatory for year 2016.
```

(continues on next page)

(continued from previous page)

```
2023-01-26 11:59:48,126 - climada_petals.entity.exposures.black_marble - INFO -␣
↪Processing country Cuba.
2023-01-26 11:59:48,987 - climada_petals.entity.exposures.black_marble - INFO -␣
↪Generating resolution of approx 2.5 km.
2023-01-26 11:59:49,129 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-01-26 11:59:49,137 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:59:49,137 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:59:49,137 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:59:49,137 - climada.entity.exposures.base - INFO - geometry not set.
2023-01-26 11:59:49,137 - climada.entity.exposures.base - INFO - centr_ not set.
2023-01-26 11:59:49,159 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-01-26 11:59:49,159 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:59:49,159 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:59:49,167 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:59:49,167 - climada.entity.exposures.base - INFO - geometry not set.
2023-01-26 11:59:49,167 - climada.entity.exposures.base - INFO - centr_ not set.
2023-01-26 11:59:49,167 - climada.entity.exposures.base - INFO - Adding sea at 5 km␣
↪resolution and 100 km distance from coast.
2023-01-26 11:59:50,567 - climada.hazard.tc_tracks - INFO - Progress: 100%
2023-01-26 11:59:50,589 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2023-01-26 12:00:02,901 - climada.util.coordinates - INFO - dist_to_coast: UTM 32616␣
↪(1/3)
2023-01-26 12:00:09,800 - climada.util.coordinates - INFO - dist_to_coast: UTM 32617␣
↪(2/3)
2023-01-26 12:00:32,928 - climada.util.coordinates - INFO - dist_to_coast: UTM 32618␣
↪(3/3)
2023-01-26 12:00:42,661 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪23083 coastal centroids.
2023-01-26 12:00:42,684 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:42,702 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪26071 coastal centroids.
2023-01-26 12:00:42,733 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:42,750 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪29579 coastal centroids.
2023-01-26 12:00:42,781 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:42,818 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪32525 coastal centroids.
2023-01-26 12:00:42,849 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:42,865 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪35241 coastal centroids.
2023-01-26 12:00:42,902 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:42,934 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪37535 coastal centroids.
2023-01-26 12:00:42,981 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,002 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪39661 coastal centroids.
2023-01-26 12:00:43,034 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,065 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪42459 coastal centroids.
2023-01-26 12:00:43,103 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,118 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪43981 coastal centroids.
2023-01-26 12:00:43,181 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,203 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
```

(continues on next page)

(continued from previous page)

```
↪45805 coastal centroids.
2023-01-26 12:00:43,250 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,266 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪47236 coastal centroids.
2023-01-26 12:00:43,319 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,334 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪48530 coastal centroids.
2023-01-26 12:00:43,381 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,404 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪48252 coastal centroids.
2023-01-26 12:00:43,435 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,466 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪47331 coastal centroids.
2023-01-26 12:00:43,504 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,535 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪45981 coastal centroids.
2023-01-26 12:00:43,582 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,604 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪44618 coastal centroids.
2023-01-26 12:00:43,635 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,667 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪43393 coastal centroids.
2023-01-26 12:00:43,704 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,720 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪42463 coastal centroids.
2023-01-26 12:00:43,767 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,782 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪41702 coastal centroids.
2023-01-26 12:00:43,820 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,836 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪41057 coastal centroids.
2023-01-26 12:00:43,869 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,900 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪40802 coastal centroids.
2023-01-26 12:00:43,936 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,951 - climada.entity.exposures.base - INFO - Matching 21923␣
↪exposures with 49817 centroids.
2023-01-26 12:00:43,951 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2023-01-26 12:00:44,005 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,005 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,021 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,021 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,021 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,036 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,052 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,052 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,052 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
```

(continues on next page)

(continued from previous page)

```
2023-01-26 12:00:44,067 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,067 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,067 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,084 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,084 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
```

```
2023-01-26 12:00:44,084 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,105 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,105 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,105 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,121 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,121 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,136 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,136 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,152 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,152 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,167 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,167 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,167 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,183 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,183 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,199 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,205 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,205 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,205 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,221 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,221 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,221 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,237 - climada.engine.impact - WARNING - The use of Impact().
```

(continues on next page)

(continued from previous page)

```
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,237 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,237 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,252 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,252 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,252 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,268 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,268 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,268 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,283 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,283 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,283 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,299 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,306 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,306 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,321 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,321 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,321 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,337 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,337 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,337 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,352 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,352 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,352 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,368 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,368 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,368 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,384 - climada.engine.impact - INFO - Generating video ./results/
↪irma_imp_fl.gif
```

```
22it [09:44, 26.57s/it]
```

10.2 Impact Functions

10.2.1 What is an impact function?

An impact function relates the percentage of damage in the exposure to the hazard intensity, also commonly referred to as a "vulnerability curve" in the modelling community. Every hazard and exposure types are characterized by an impact function.

10.2.2 What is the difference between `ImpactFunc` and `ImpactFuncSet`?

An `ImpactFunc` is a class for a single impact function. E.g. a function that relates the percentage of damage of a reinforced concrete building (exposure) to the wind speed of a tropical cyclone (hazard intensity).

An `ImpactFuncSet` class is a container that contains multiple `ImpactFunc`. For instance, there are 100 `ImpactFunc` represent 100 types of buildings exposed to tropical cyclone's wind damage. These 100 `ImpactFunc` are all gathered in an `ImpactFuncSet`.

10.2.3 What does an `ImpactFunc` look like in CLIMADA?

The `ImpactFunc` class requires users to define the following attributes.

Mandatory attributes	Data Type	Description
haz_type	(str)	Hazard type acronym (e.g. 'TC')
id	(int or str)	Unique id of the impact function. Exposures of the same type will refer to the same impact function id
name	(str)	Name of the impact function
intensity	(np.array)	Intensity values
intensity_unit	(str)	Unit of the intensity
mdd	(np.array)	Mean damage (impact) degree for each intensity (numbers in [0,1])
paa	(np.array)	Percentage of affected assets (exposures) for each intensity (numbers in [0,1])

Users may use `ImpactFunc.check()` to check that the attributes have been set correctly. The mean damage ratio `mdr` (mdr=mdd*paa) is calculated by the method `ImpactFunc.calc_mdr()`.

10.2.4 What does an `ImpactFuncSet` look like in CLIMADA?

The `ImpactFuncSet` class contains all the `ImpactFunc` classes. Users are not required to define any attributes in `ImpactFuncSet`.

To add an `ImpactFunc` into an `ImpactFuncSet`, simply use the method `ImpactFuncSet.append(ImpactFunc)`. If the users only has one impact function, they should generate an `ImpactFuncSet` that contains one impact function. `ImpactFuncSet` is to be used in the *impact calculation*.

Attributes	Data Type	Description
_data	(dict)	Contains `ImpactFunc` classes. Not suppossed to be directly accessed. Use the class methods instead.

10.2.5 Structure of the tutorial

Part 1: Defining `ImpactFunc` from your own data

Part 2: Loading `ImpactFunc` from CLIMADA in-built impact functions

Part 3: Add `ImpactFunc` into the container `ImpactFuncSet`

Part 4: Read and write `ImpactFuncSet` into Excel sheets

Part 5: Loading `ImpactFuncSet` from CLIMADA in-built impact functions

10.2.6 Part 1: Defining `ImpactFunc` from your own data

The essential attributes are listed in the table above. The following example shows you how to define an `ImpactFunc` from scratch, and using the method `ImpactFunc.calc_mdr()` to calculate the mean damage ratio.

Generate a dummy impact function from scratch.

Here we generate an impact function with random dummy data for illustrative reasons. Assuming this impact function is a function that relates building damage to tropical cyclone (TC) wind, with an arbitrary id 3.

```
import numpy as np
from climada.entity import ImpactFunc

# We initialise a dummy ImpactFunc for tropical cyclone wind damage to building.
# Giving the ImpactFunc an arbitrary id 3.
haz_type = "TC"
id = 3
name = "TC building damage"
# provide unit of the hazard intensity
intensity_unit = "m/s"
# provide values for the hazard intensity, mdd, and paa
intensity = np.linspace(0, 100, num=15)
mdd = np.concatenate((np.array([0]), np.sort(np.random.rand(14))), axis=0)
paa = np.concatenate((np.array([0]), np.sort(np.random.rand(14))), axis=0)
imp_fun = ImpactFunc(
    id=id,
    name=name,
    intensity_unit=intensity_unit,
    haz_type=haz_type,
    intensity=intensity,
    mdd=mdd,
    paa=paa,
)

# check if the all the attributes are set correctly
imp_fun.check()
```

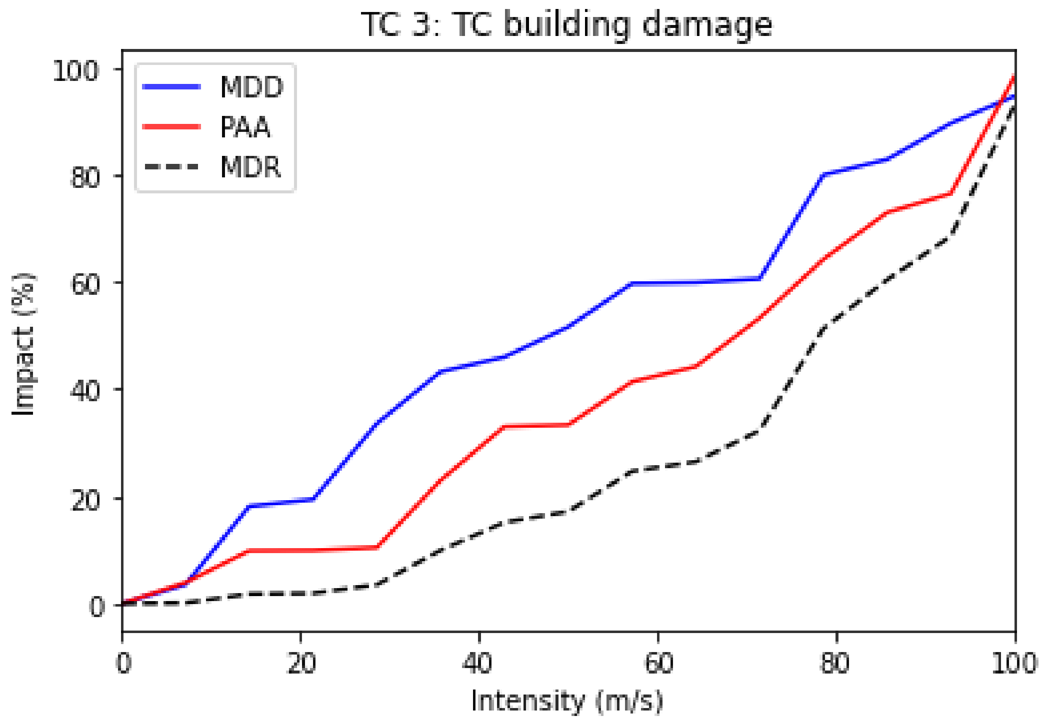
```
# Calculate the mdr at hazard intensity 18.7 m/s
print('Mean damage ratio at intensity 18.7 m/s: ', imp_fun.calc_mdr(18.7))
```

```
Mean damage ratio at intensity 18.7 m/s:  0.01878041941423081
```

Visualise the Impact function

The method `plot()` uses the matplotlib's axes plot function to visualise the impact function. It returns a figure and axes, which can be modified by users.

```
# plot impact function
imp_fun.plot();
```



10.2.7 Part 2: Loading impact functions from CLIMADA in-built impact functions

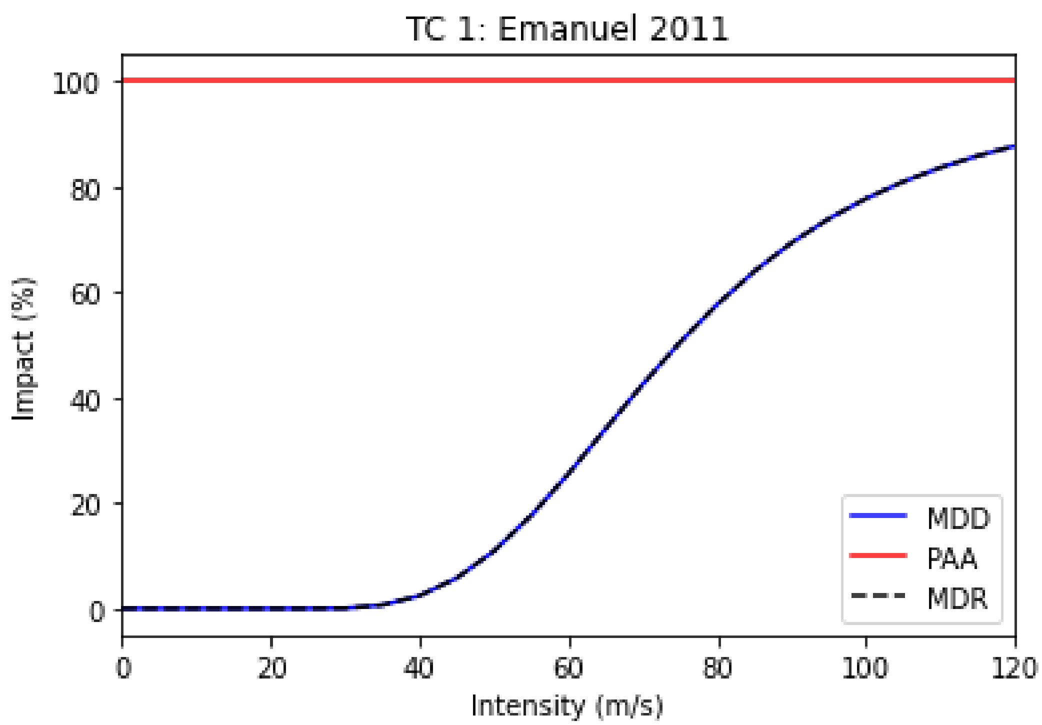
In CLIMADA there is several defined impact functions that users can directly load and use them. However, users should be aware of the applications of the impact functions to types of assets, reading the background references of the impact functions are strongly recommended. Currently available perils include tropical cyclones, river floods, European windstorm, crop yield, and drought.Continuous updates of perils are available. Here we use the impact function of tropical cyclones as an example.

Loading CLIMADA in-built impact function for tropical cyclones

`ImpfTropCyclone` is a derivated class of `ImpactFunc`. This in-built impact function estimates the insured property damages by tropical cyclone wind in USA, following the reference paper Emanuel (2011).

To generate this impact function, method `set_emanual_usa()` is used.

```
from climada.entity import ImpfTropCyclone

# Here we generate the impact function for TC damage using the formula of Emanuel 2011
impFunc_emanuel_usa = ImpfTropCyclone.from_emanuel_usa()
# plot the impact function
impFunc_emanuel_usa.plot();
```



10.2.8 Part 3: Add `ImpactFunc` into the container `ImpactFuncSet`

`ImpactFuncSet` is a container of multiple `ImpactFunc`, it is part of the arguments in `ImpactCalc.impact()` (see *the impact tutorial*).

Here we generate 2 arbitrary impact functions and add them into an `ImpactFuncSet` class. To add them into the container, simply use the method `ImpactFuncSet.append(ImpactFunc)`.

```
import numpy as np
import matplotlib.pyplot as plt
from climada.entity import ImpactFunc, ImpactFuncSet

# generate the 1st arbitrary impact function
haz_type = "TC"
id = 1
name = "TC Default Damage Function"
intensity_unit = "m/s"
intensity = np.linspace(0, 100, num=10)
mdd = np.concatenate((np.array([0]), np.sort(np.random.rand(9))), axis=0)
paa = np.concatenate((np.array([0]), np.sort(np.random.rand(9))), axis=0)
imp_fun_1 = ImpactFunc(
    id=id,
    name=name,
    intensity_unit=intensity_unit,
    haz_type=haz_type,
    intensity=intensity,
    mdd=mdd,
    paa=paa,
)
```

(continues on next page)

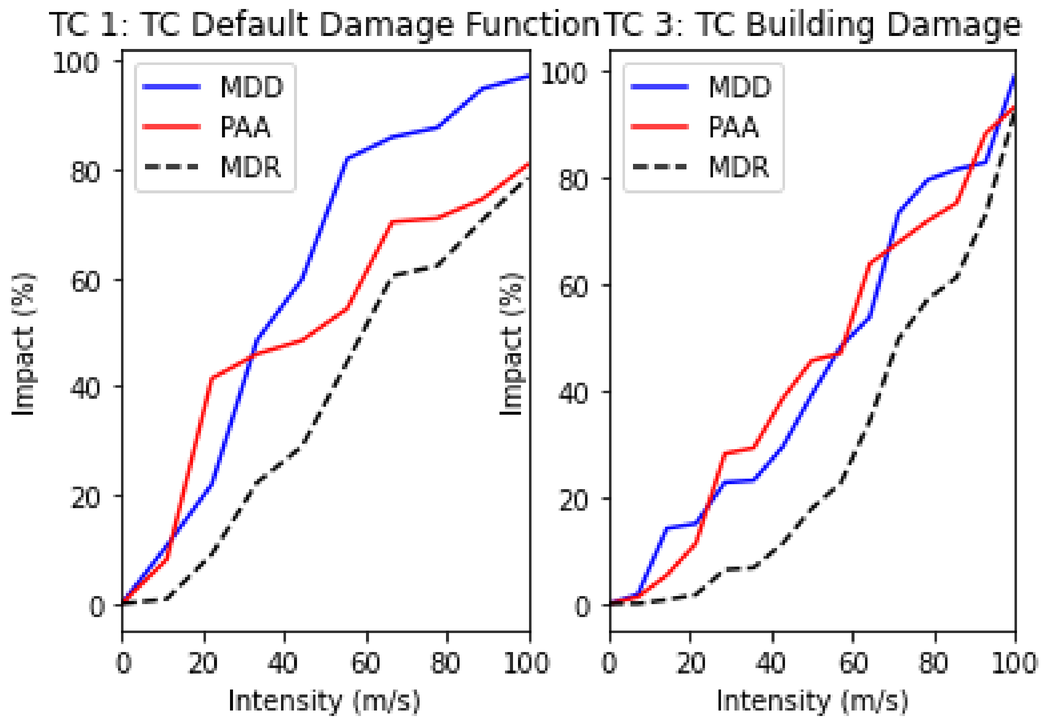
(continued from previous page)

```
imp_fun_1.check()

# generate the 2nd arbitrary impact function
haz_type = "TC"
id = 3
name = "TC Building Damage"
intensity_unit = "m/s"
intensity = np.linspace(0, 100, num=15)
mdd = np.concatenate((np.array([0]), np.sort(np.random.rand(14))), axis=0)
paa = np.concatenate((np.array([0]), np.sort(np.random.rand(14))), axis=0)
imp_fun_3 = ImpactFunc(
    id=id,
    name=name,
    intensity_unit=intensity_unit,
    haz_type=haz_type,
    intensity=intensity,
    mdd=mdd,
    paa=paa,
)
imp_fun_3.check()

# add the 2 impact functions into ImpactFuncSet
imp_fun_set = ImpactFuncSet([imp_fun_1, imp_fun_3])
```

Plotting all the impact functions in an `ImpactFuncSet`

The method `plot()` in `ImpactFuncSet` also uses the the matplotlib's axes plot function to visualise the impact functions, returning a figure with all the subplots of impact functions. Users may modify these plots.

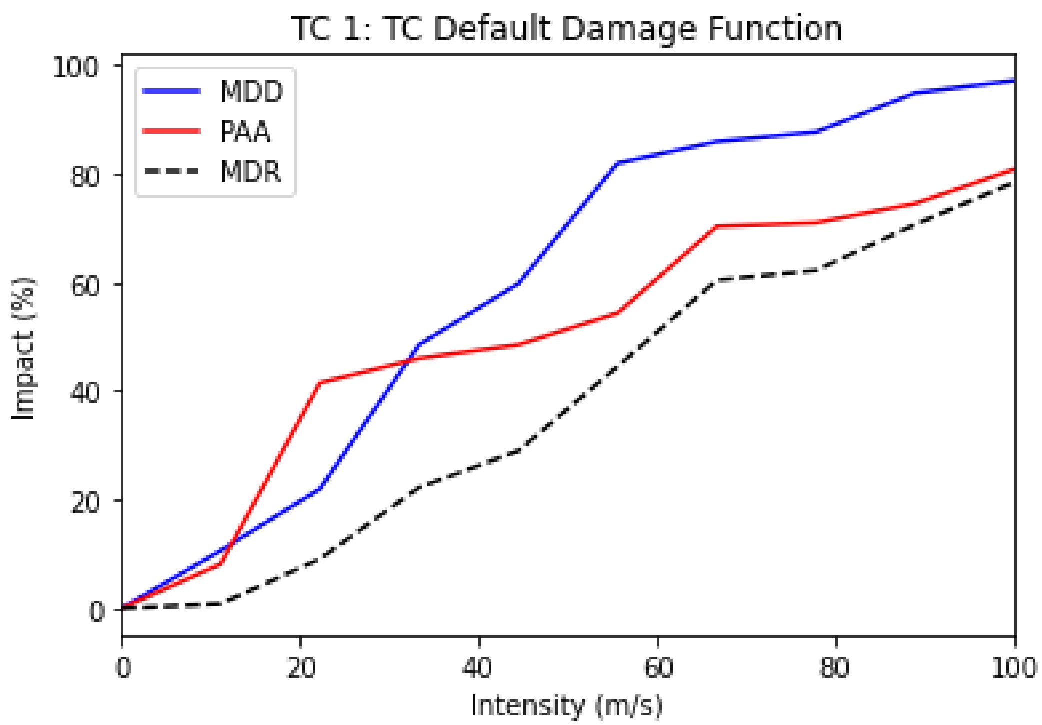
```
# plotting all the impact functions in impf_set
axes = imp_fun_set.plot()
```



Retrieving an impact function from the `ImpactFuncSet`

User may want to retrive a particular impact function from `ImpactFuncSet`. Using the method `get_func(haz_type, id)`, it returns an `ImpactFunc` class of the desired impact function. Below is an example of extracting the TC impact function with id 1, and using `plot()` to visualise the function.

```
# extract the TC impact function with id 1
impf_tc_1 = imp_fun_set.get_func('TC', 1)
# plot the impact function
impf_tc_1.plot();
```

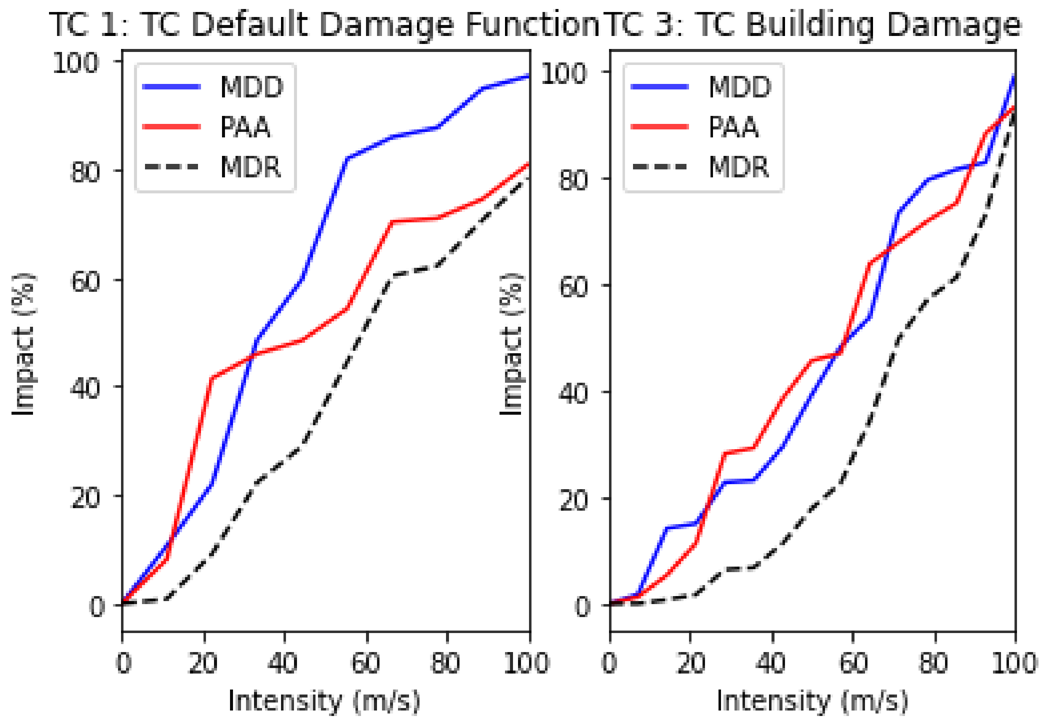


Removing an impact function from the `ImpactFuncSet`

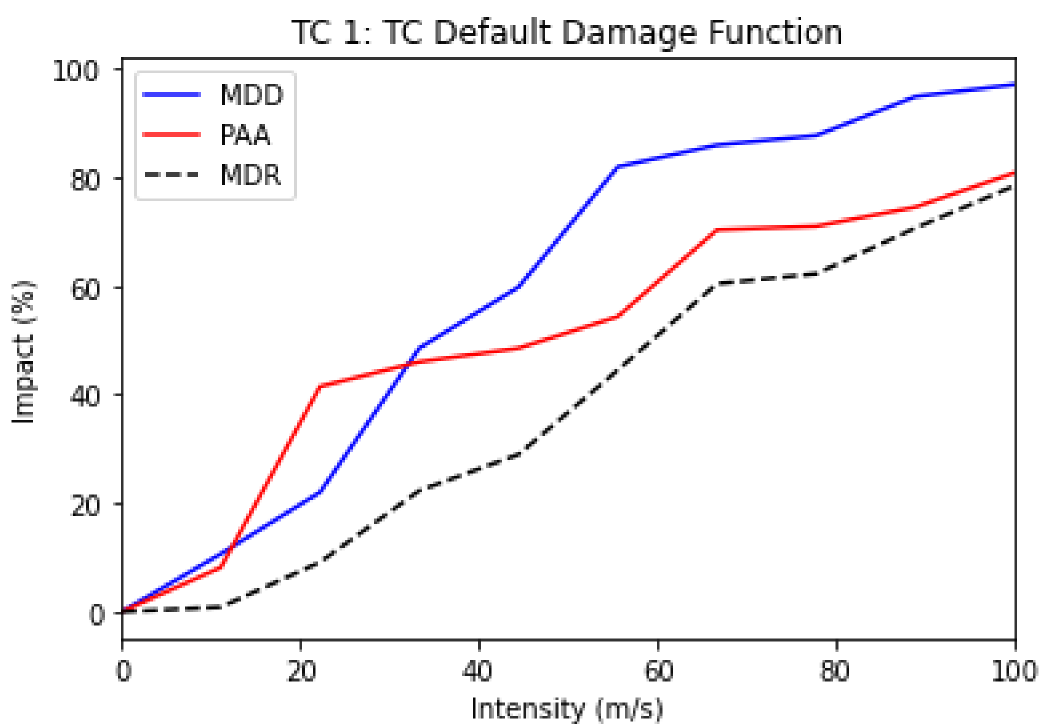
If there is an unwanted impact function from the `ImpactFuncSet`, we may remove it using the method `remove_func(haz_type, id)` to remove it from the set.

For example, in the previous generated impact function set `imp_fun_set` contains an unwanted TC impact function with id 3, we might thus would like to remove that from the set.

```
# first plotting all the impact functions in the impact function set to see what is␣
↪in there:
imp_fun_set.plot();
```




```
# removing the TC impact function with id 3
imp_fun_set.remove_func('TC', 3)
# plot all the remaining impact functions in imp_fun_set
imp_fun_set.plot();
```



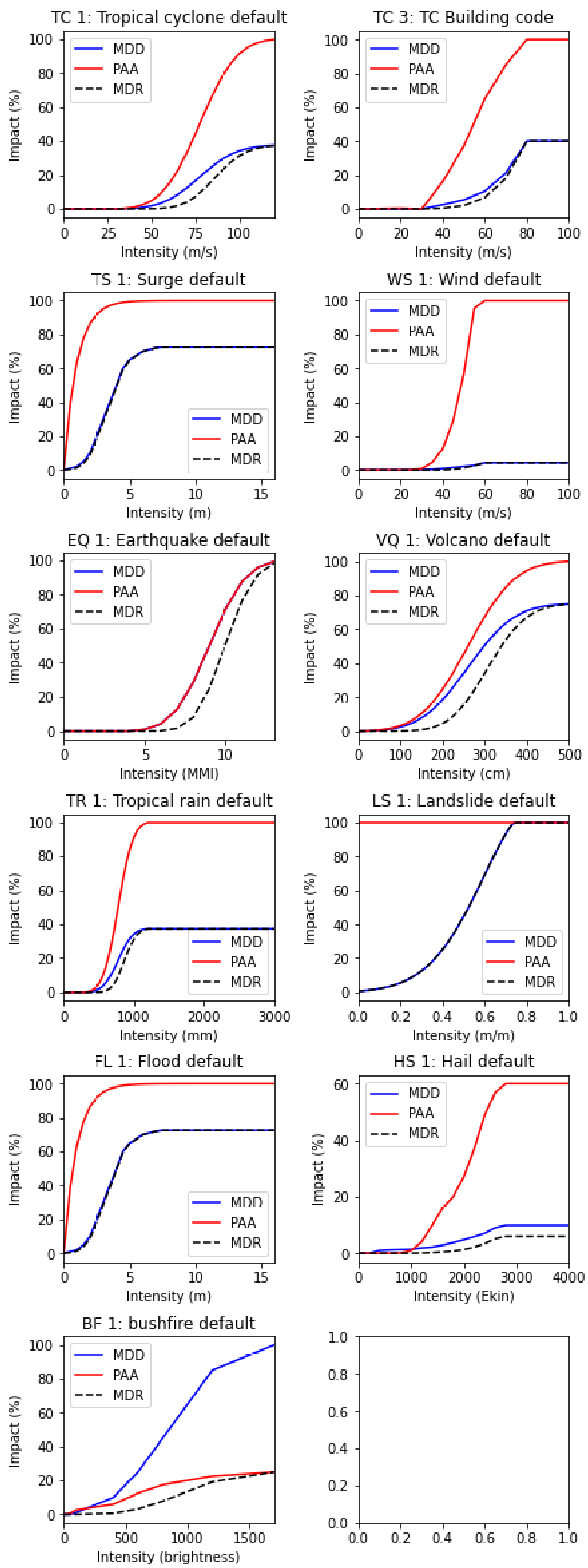
10.2.9 Part 4: Read and write `ImpactFuncSet` into Excel sheets

Users may load impact functions to an `ImpactFuncSet` class from an excel sheet, or to write the `ImpactFuncSet` into an excel sheet. This section will give an example of how to do it.

Reading impact functions from an Excel file

Impact functions defined in an excel file following the template provided in sheet `impact_functions` of `climada_python/climada/data/system/entity_template.xlsx` can be ingested directly using the method `from_excel()`.

```
from climada.entity import ImpactFuncSet
from climada.util import ENT_TEMPLATE_XLS
import matplotlib.pyplot as plt

# provide absolute path of the input excel file
file_name = ENT_TEMPLATE_XLS
# fill ImpactFuncSet from Excel file
imp_set_xlsx = ImpactFuncSet.from_excel(file_name)

# plot all the impact functions from the ImpactFuncSet
imp_set_xlsx.plot()
# adjust the plots
plt.subplots_adjust(right=1., top=4., hspace=0.4, wspace=0.4)
```


TC 1: Tropical cyclone default
Impact (%)
Intensity (m/s)
MDD
PAA
MDR
TC 3: TC Building code
Impact (%)
Intensity (m/s)
MDD
PAA
MDR
TS 1: Surge default
Impact (%)
Intensity (m)
MDD
PAA
MDR
WS 1: Wind default
Impact (%)
Intensity (m/s)
MDD
PAA
MDR
EQ 1: Earthquake default
Impact (%)
Intensity (MMI)
MDD
PAA
MDR
VQ 1: Volcano default
Impact (%)
Intensity (cm)
MDD
PAA
MDR
TR 1: Tropical rain default
Impact (%)
Intensity (mm)
MDD
PAA
MDR
LS 1: Landslide default
Impact (%)
Intensity (m/m)
MDD
PAA
MDR
FL 1: Flood default
Impact (%)
Intensity (m)
MDD
PAA
MDR
HS 1: Hail default
Impact (%)
Intensity (Ekin)
MDD
PAA
MDR
BF 1: bushfire default
Impact (%)
Intensity (brightness)
MDD
PAA
MDR

Write impact functions

Users may write the impact functions in Excel format using `write_excel()` method.

```
# write imp_set_xlsx into an excel file
imp_set_xlsx.write_excel('tutorial_impf_set.xlsx')
```

Alternative saving format

Alternatively, users may also save the impact functions into pickle format, using CLIMADA in-built function `save()`.

```
from climada.util.save import save

# this generates a results folder in the current path and stores the output there
save('tutorial_impf_set.p', imp_set_xlsx)
```

```
2022-03-28 20:08:56,846 - climada.util.save - INFO - Written file /mnt/c/users/yyljy/
↪documents/climada_main/doc/tutorial/results/tutorial_impf_set.p
```

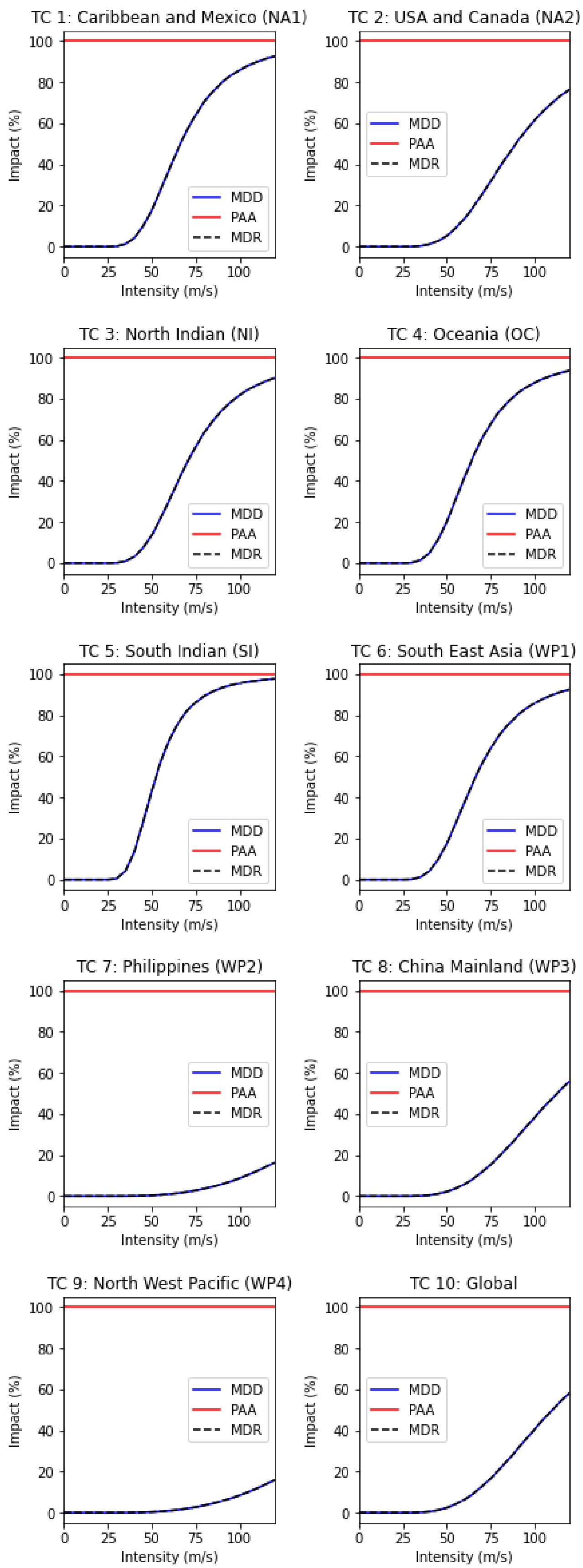
10.2.10 Part 5: Loading `ImpactFuncSet` from CLIMADA in-built impact functions

Similar to *Part 3*, some of the impact functions are available as `ImpactFuncSet` classes. Users may load them from the CLIMADA modules.

Here we use the example of the calibrated impact functions of TC wind damages per region to property damages, following the reference Eberenz et al. (2021). Method `from_calibrated_regional_ImpfSet()` returns a set of default calibrated impact functions for TC for different regions.

```
from climada.entity.impact_funcs.trop_cyclone import ImpfSetTropCyclone
import matplotlib.pyplot as plt

# generate the default calibrated TC impact functions for different regions
imp_fun_set_TC = ImpfSetTropCyclone.from_calibrated_regional_ImpfSet()

# plot all the impact functions
imp_fun_set_TC.plot()
# adjust the plots
plt.subplots_adjust(right=1., top=4., hspace=0.4, wspace=0.4)
```

```
/tmp/ipykernel_1009/2983082256.py:10: UserWarning: Tight layout not applied. tight_
↪layout cannot make axes height small enough to accommodate all axes decorations.
  plt.tight_layout()
```


TC 1: Caribbean and Mexico (NA1)
TC 2: USA and Canada (NA2)
TC 3: North Indian (NI)
TC 4: Oceania (OC)
TC 5: South Indian (SI)
TC 6: South East Asia (WP1)
TC 7: Philippines (WP2)
TC 8: China Mainland (WP3)
TC 9: North West Pacific (WP4)
TC 10: Global
Impact (%)
Intensity (m/s)
0
20
40
60
80
100
0
25
50
75
100
MDD
PAA
MDR

10.3 Adaptation Measures

Adaptation measures are defined by parameters that alter the exposures, hazard or impact functions. Risk transfer options are also considered. Single measures are defined in the `Measure` class, which can be aggregated to a `MeasureSet`.

10.3.1 Measure class

A measure is characterized by the following attributes:

Related to measure's description:

- name (str): name of the action
- haz_type (str): related hazard type (peril), e.g. TC
- color_rgb (np.array): integer array of size 3. Gives color code of this measure in RGB
- cost (float): discounted cost (in same units as assets). Needs to be provided by the user. See the example provided in `climada_python/climada/data/system/entity_template.xlsx` sheets `_measures_details` and `_discounting_sheet` to see how the discounting is done.

Related to a measure's impact:

- hazard_set (str): file name of hazard to use
- hazard_freq_cutoff (float): hazard frequency cutoff
- exposure_set (str): file name of exposure to use
- hazard_inten_imp (tuple): parameter a and b of hazard intensity change
- mdd_impact (tuple): parameter a and b of the impact over the mean damage degree
- paa_impact (tuple): parameter a and b of the impact over the percentage of affected assets
- imp_fun_map (str): change of impact function id, e.g. '1to3'
- exp_region_id (int): region id of the selected exposures to consider ALL the previous parameters
- risk_transf_attach (float): risk transfer attachment. Applies to the whole exposure.
- risk_transf_cover (float): risk transfer cover. Applies to the whole exposure.

Paramters description:

`hazard_set` and `exposures_set` provide the file names in h5 format (generated by CLIMADA) of the hazard and exposures to use as a result of the implementation of the measure. These might be further modified when applying the other parameters.

`hazard_inten_imp`, `mdd_impact` and `paa_impact` transform the impact functions linearly as follows:

```
intensity = intensity*hazard_inten_imp[0] + hazard_inten_imp[1]
mdd = mdd*mdd_impact[0] + mdd_impact[1]
paa = paa*paa_impact[0] + paa_impact[1]
```

`hazard_freq_cutoff` modifies the hazard by putting 0 intensities to the events whose impact exceedance frequency are greater than `hazard_freq_cutoff`.

`imp_fun_map` indicates the ids of the impact function to replace and its replacement. The `impf_XX` variable of `Exposures` with the affected impact function id will be correspondingly modified (`XX` refers to the `haz_type` of the measure).

`exp_region_id` will apply all the previous changes only to the `region_id` indicated. This means that only the exposures with that `region_id` and the hazard's centroids close to them will be modified with the previous changes, the other regions will remain unaffected to the measure.

`risk_transf_attach` and `risk_transf_cover` are the deductible and coverage of any event to happen.

Methods description:

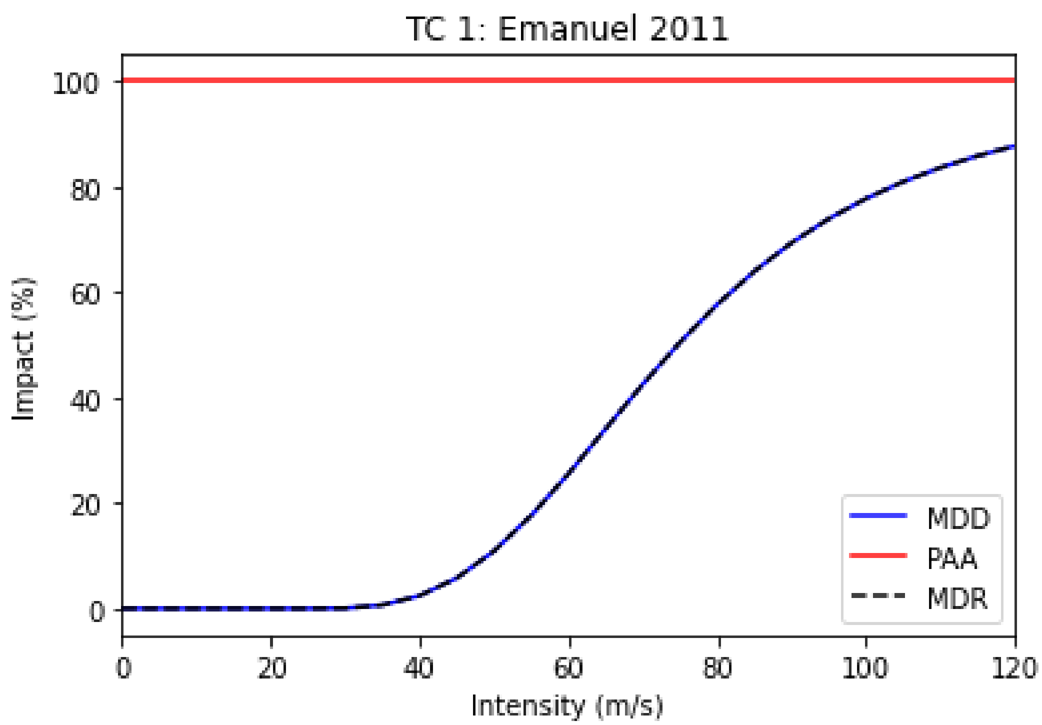
The method `check()` validates the attibutes. `apply()` applies the measure to a given exposure, impact function and hazard, returning their modified values. The parameters related to insurability (risk_transf_attach and risk_transf_cover) affect the resulting impact and are therefore not applied in the `apply()` method yet.

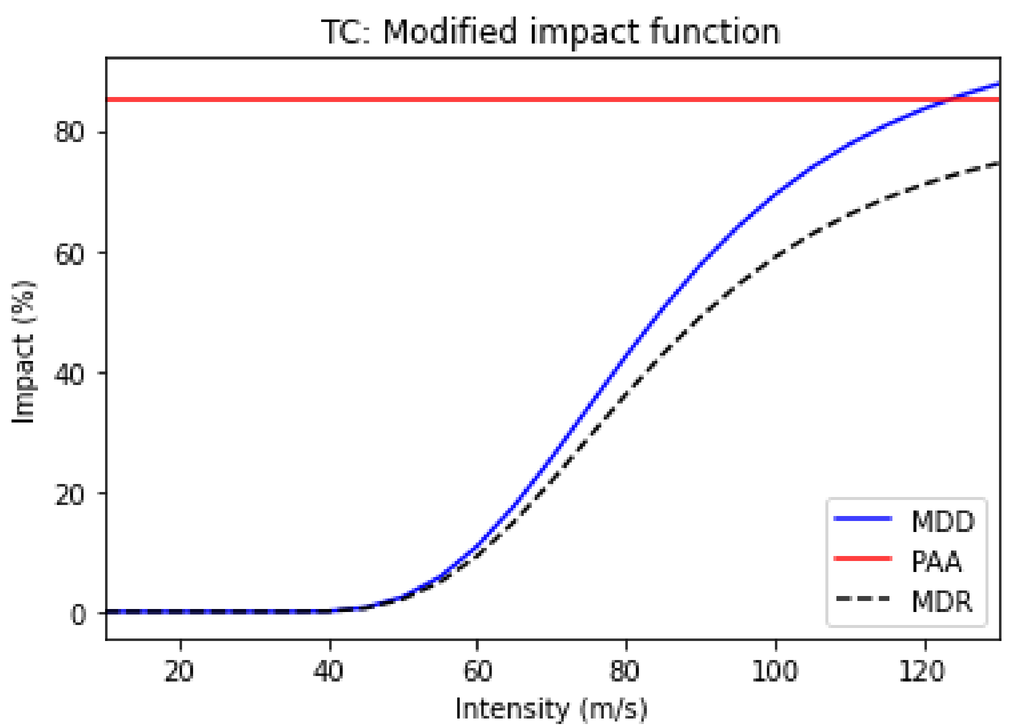
`calc_impact()` calls to `apply()`, applies the insurance parameters and returns the final impact and risk transfer of the measure. This method is called from the `CostBenefit` class.

The method `apply()` allows to visualize the effect of a measure. Here are some examples:

```
# effect of mdd_impact, paa_impact, hazard_inten_imp
%matplotlib inline
import numpy as np
from climada.entity import ImpactFuncSet, ImpfTropCyclone, Exposures
from climada.entity.measures import Measure
from climada.hazard import Hazard

# define measure
meas = Measure(
    name='Mangrove',
    haz_type='TC',
    color_rgb=np.array([1, 1, 1]),
    cost=500000000,
    mdd_impact=(1, 0),
    paa_impact=(1, -0.15),
    hazard_inten_imp=(1, -10), # reduces intensity by 10
)

# impact functions
impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_all = ImpactFuncSet([impf_tc])
impf_all.plot();

# dummy Hazard and Exposures
haz = Hazard('TC') # this measure does not change hazard
exp = Exposures() # this measure does not change exposures

# new impact functions
new_exp, new_impfs, new_haz = meas.apply(exp, impf_all, haz)
axes = new_impfs.plot();
axes.set_title('TC: Modified impact function')
```

```
Text(0.5, 1.0, 'TC: Modified impact function')
```






```
# effect of hazard_freq_cutoff
import numpy as np
from climada.entity import ImpactFuncSet, ImpfTropCyclone, Exposures
from climada.entity.measures import Measure
from climada.hazard import Hazard
from climada.engine import ImpactCalc
```

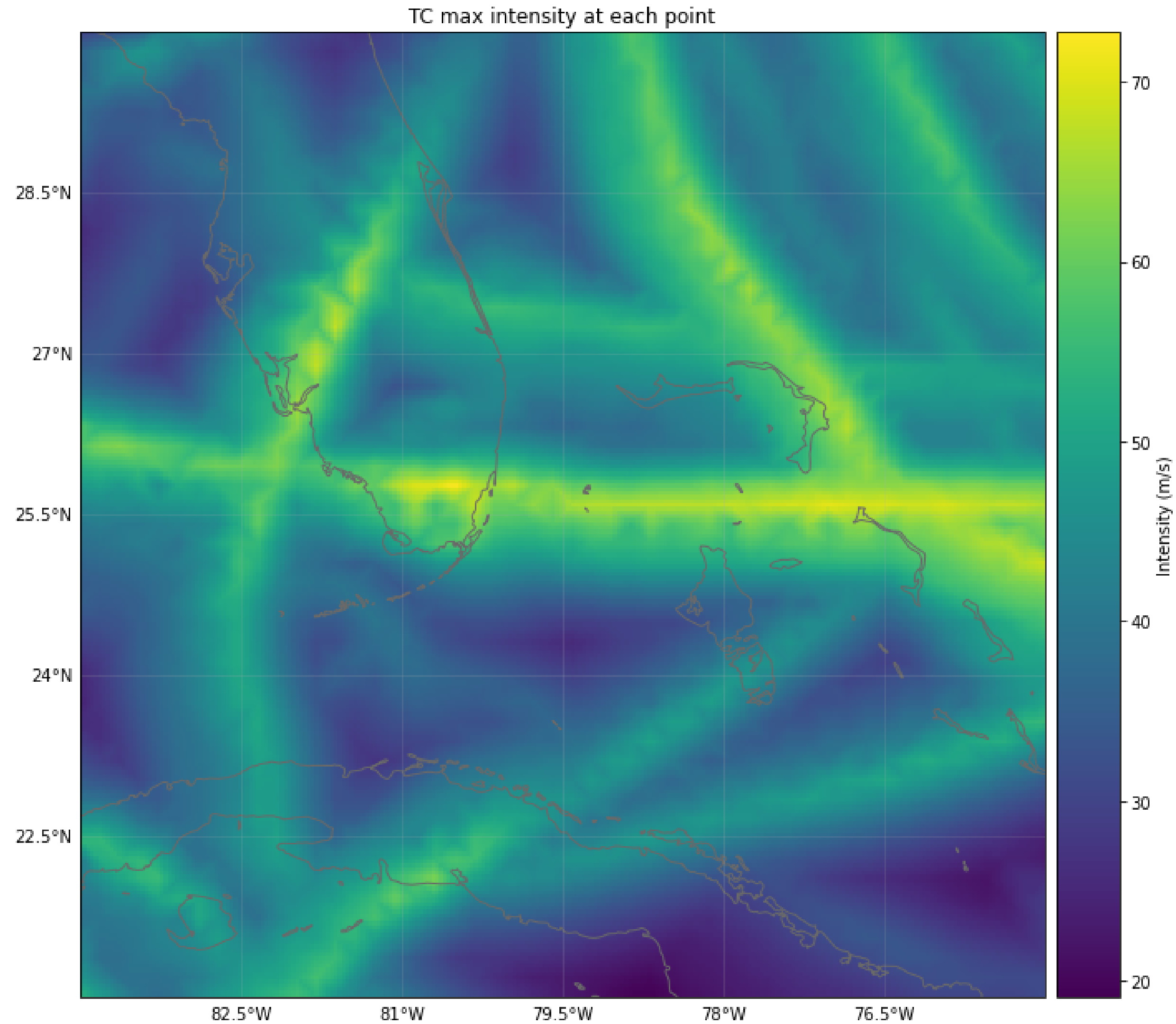
(continues on next page)

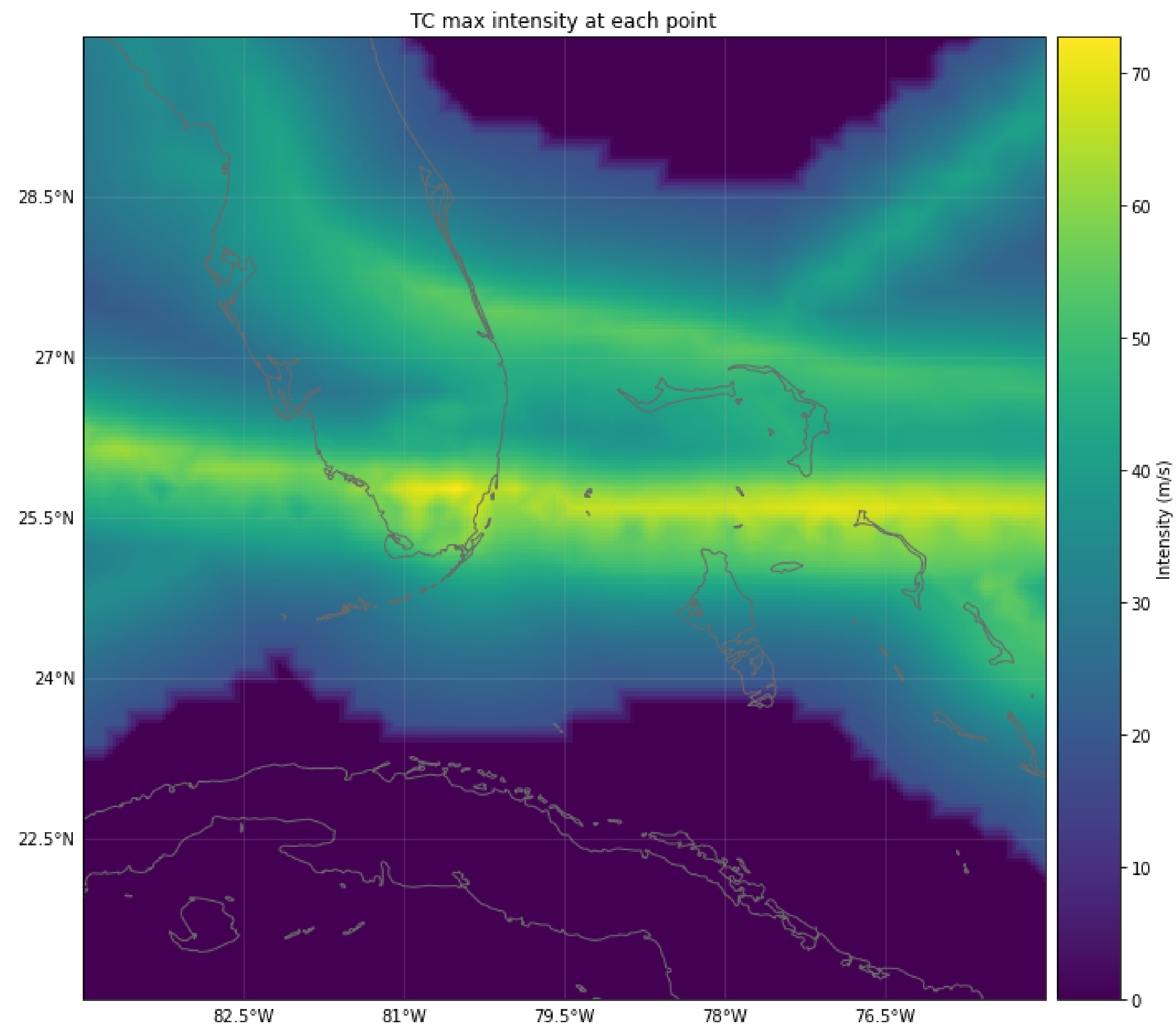
(continued from previous page)

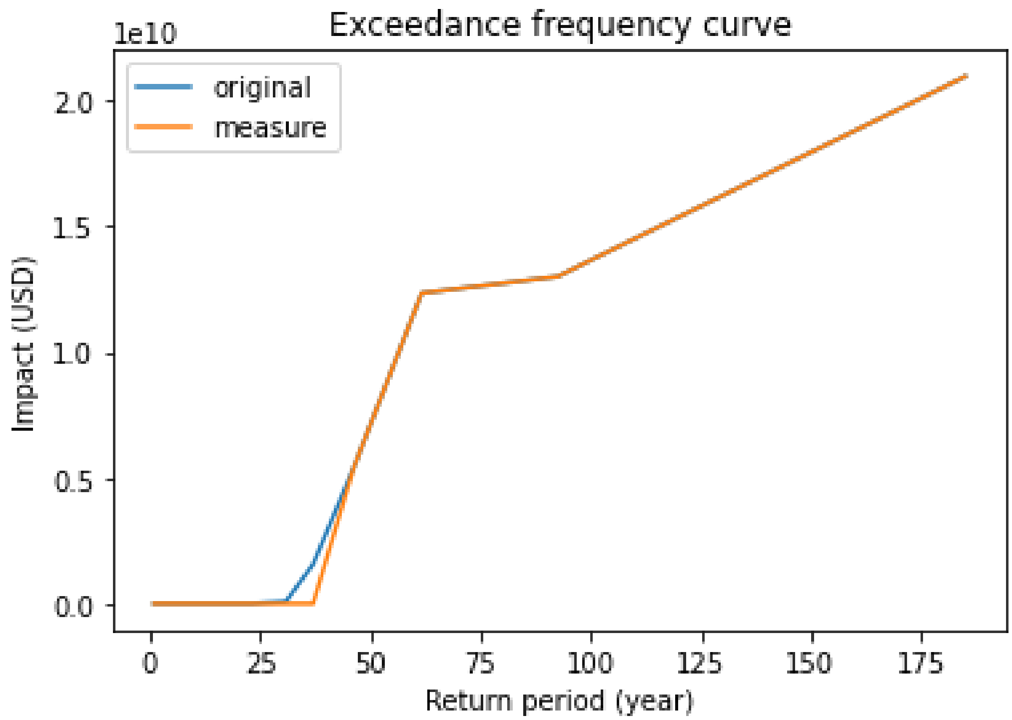
```
from climada.util import HAZ_DEMO_H5, EXP_DEMO_H5

# define measure
meas = Measure(
    name='Mangrove',
    haz_type='TC',
    color_rgb=np.array([1, 1, 1]),
    cost=500000000,
    hazard_freq_cutoff=0.0255,
)

# impact functions
impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_all = ImpactFuncSet([impf_tc])

# Hazard
haz = Hazard.from_hdf5(HAZ_DEMO_H5)
haz.check()

# Exposures
exp = Exposures.from_hdf5(EXP_DEMO_H5)
exp.check()

# new hazard
new_exp, new_impfs, new_haz = meas.apply(exp, impf_all, haz)
# if you look at the maximum intensity per centroid: new_haz does not contain the␣
↪event with smaller impact (the most frequent)
haz.plot_intensity(0);
new_haz.plot_intensity(0);
# you might also compute the exceedance frequency curve of both hazard
imp = ImpactCalc(exp, impf_all, haz).impact()
ax = imp.calc_freq_curve().plot(label='original');

new_imp = ImpactCalc(new_exp, new_impfs, new_haz).impact()
new_imp.calc_freq_curve().plot(axis=ax, label='measure'); # the damages for events␣
↪with return periods > 1/0.0255 ~ 40 are 0
```

```
$CONDA_PREFIX/lib/python3.8/site-packages/pyproj/crs/crs.py:68: FutureWarning: '+init=
↪<authority>:<code>' syntax is deprecated. '<authority>:<code>' is the preferred␣
↪initialization method. When making the change, be mindful of axis order changes:␣
↪https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-in-proj-6
  return _prepare_from_string(" ".join(pjargs))
```

```
<matplotlib.legend.Legend at 0x7f433756d970>
```


TC max intensity at each point
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
70
60
50
40
30
20
Intensity (m/s)


TC max intensity at each point
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
70
60
50
40
30
20
10
0
Intensity (m/s)




```
# effect of exp_region_id
import numpy as np
from climada.entity import ImpactFuncSet, ImpfTropCyclone, Exposures
from climada.entity.measures import Measure
from climada.hazard import Hazard
from climada.engine import ImpactCalc

from climada.util import HAZ_DEMO_H5, EXP_DEMO_H5

# define measure
meas = Measure(
    name='Building code',
    haz_type='TC',
    color_rgb=np.array([1, 1, 1]),
    cost=500000000,
    hazard_freq_cutoff=0.00455,
    exp_region_id=[1], # apply measure to points close to exposures with region_id=1
)

# impact functions
impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_all = ImpactFuncSet([impf_tc])

# Hazard
haz = Hazard.from_hdf5(HAZ_DEMO_H5)
haz.check()

# Exposures
exp = Exposures.from_hdf5(EXP_DEMO_H5)
#exp['region_id'] = np.ones(exp.shape[0])
exp.check()
# all exposures have region_id=1
```

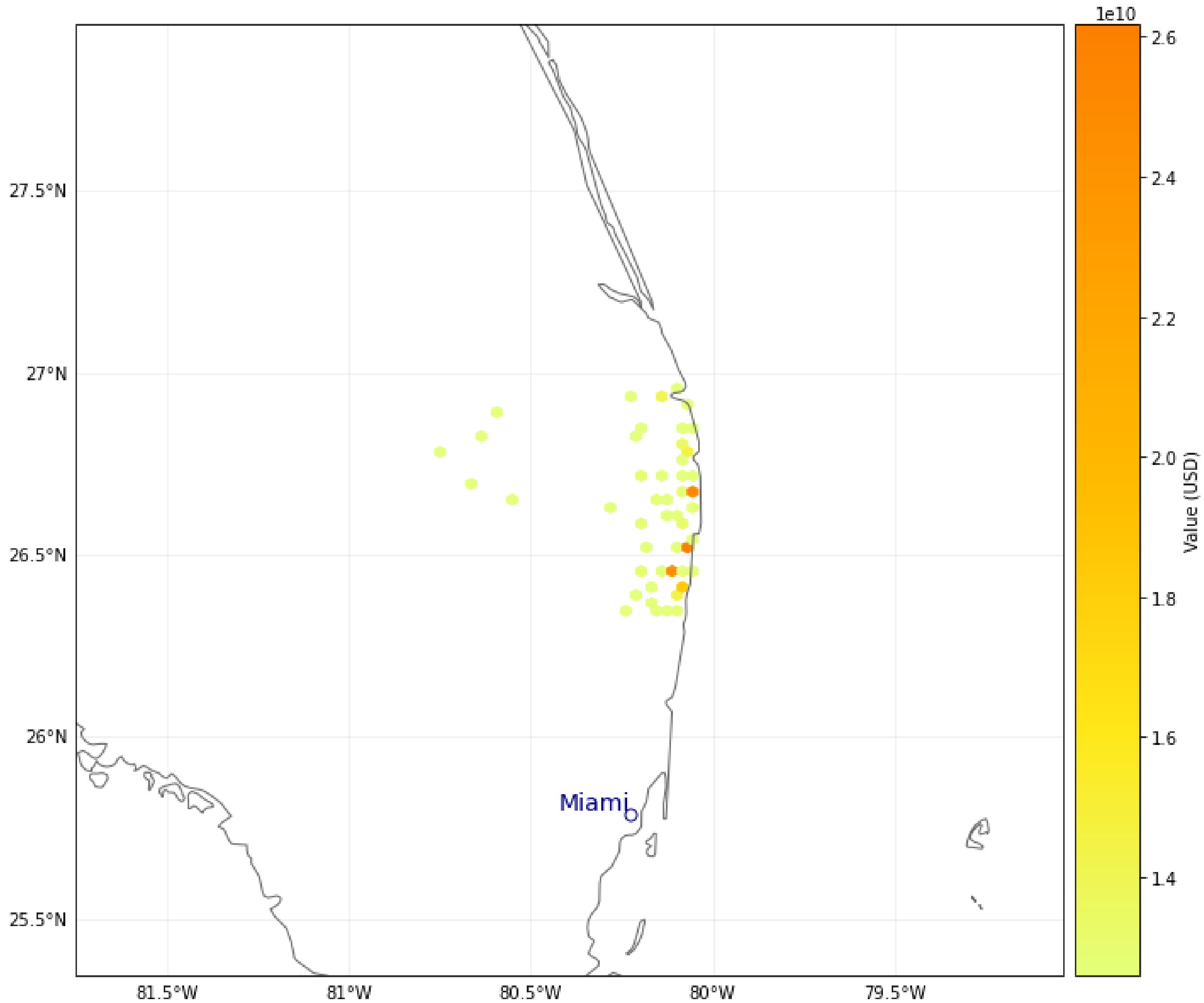
(continues on next page)

(continued from previous page)

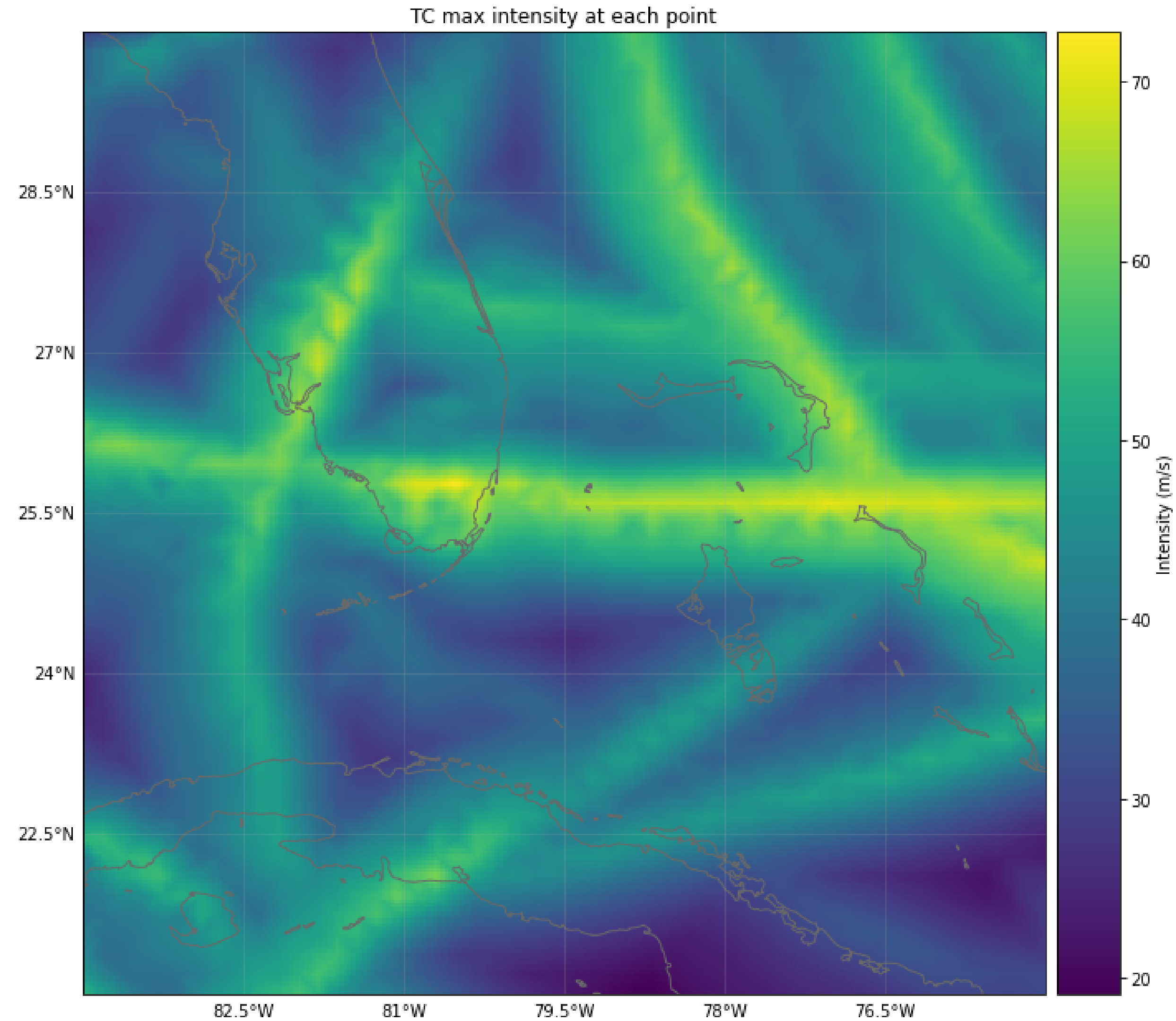
```
exp.plot_hexbin(buffer=1.0)

# new hazard
new_exp, new_impfs, new_haz = meas.apply(exp, impf_all, haz)
# the cutoff has been apllied only in the region of the exposures
haz.plot_intensity(0)
new_haz.plot_intensity(0)

# the exceddance frequency has only been computed for the selected exposures before␣
↪doing the cutoff.
# since we have removed the hazard of the places with exposure, the new exceedance␣
↪frequency curve is zero.
imp = ImpactCalc(exp, impf_all, haz).impact()
imp.calc_freq_curve().plot()

new_imp = ImpactCalc(new_exp, new_impfs, new_haz).impact()
new_imp.calc_freq_curve().plot();
```

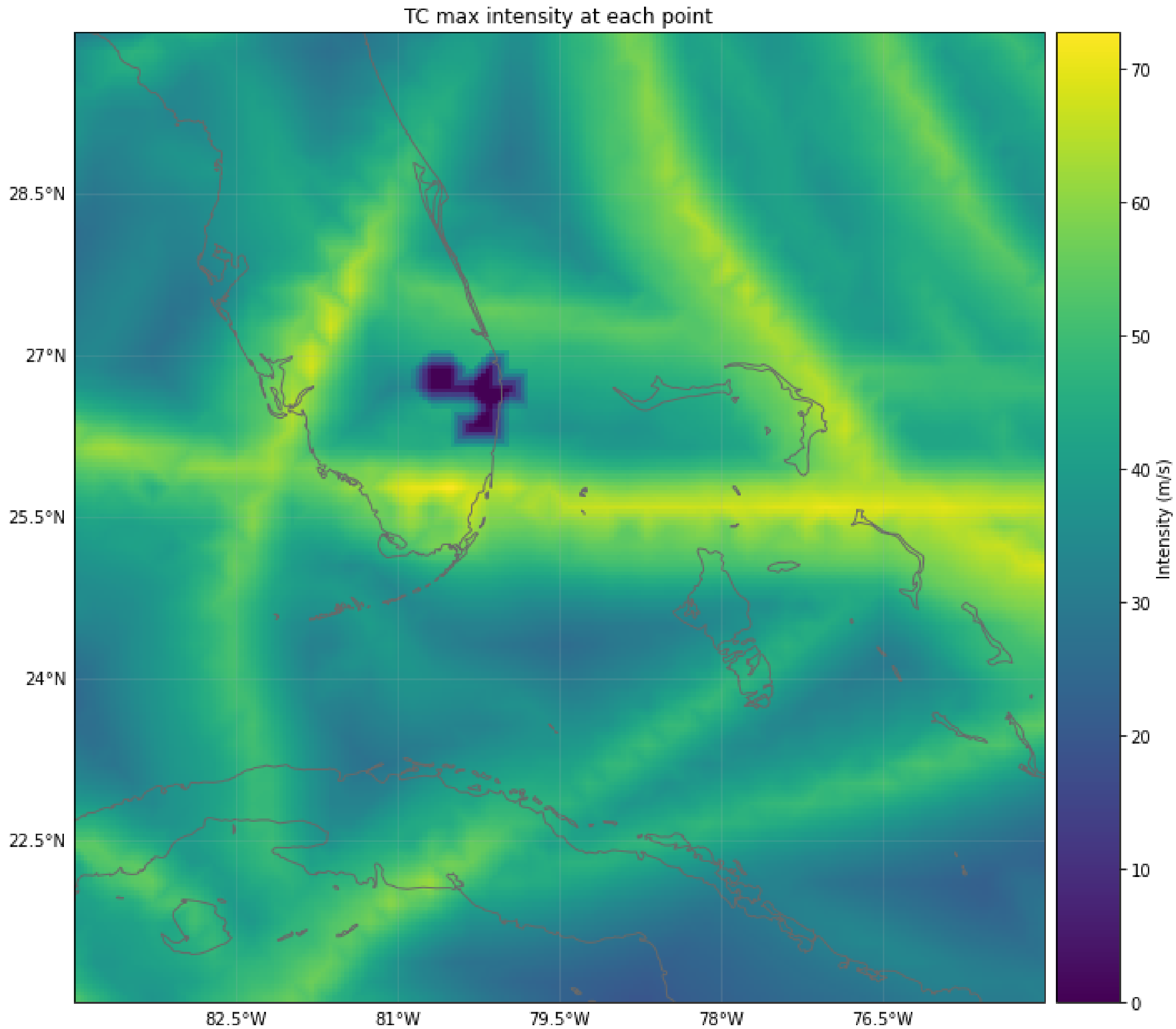
```
$CONDA_PREFIX/lib/python3.8/site-packages/pyproj/crs/crs.py:68: FutureWarning: '+init=
↪<authority>:<code>' syntax is deprecated. '<authority>:<code>' is the preferred␣
↪initialization method. When making the change, be mindful of axis order changes:␣
↪https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-in-proj-6
  return _prepare_from_string(" ".join(pjargs))
```

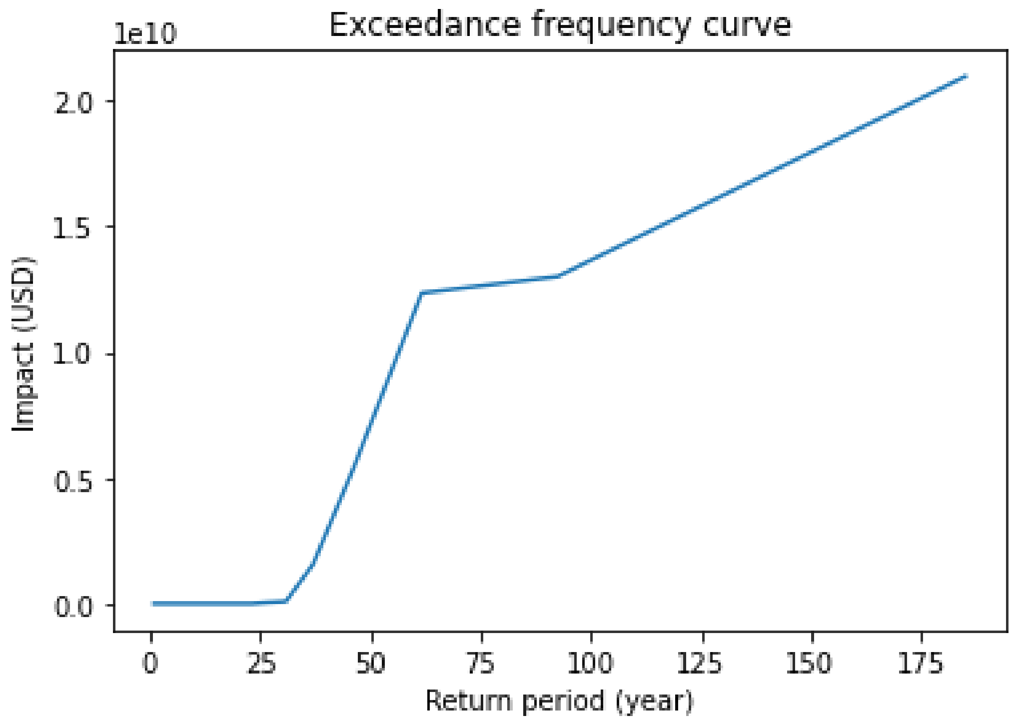
```
<AxesSubplot:title={'center':'Exceedance frequency curve'}, xlabel='Return period␣
↪(year)', ylabel='Impact (USD)'>
```

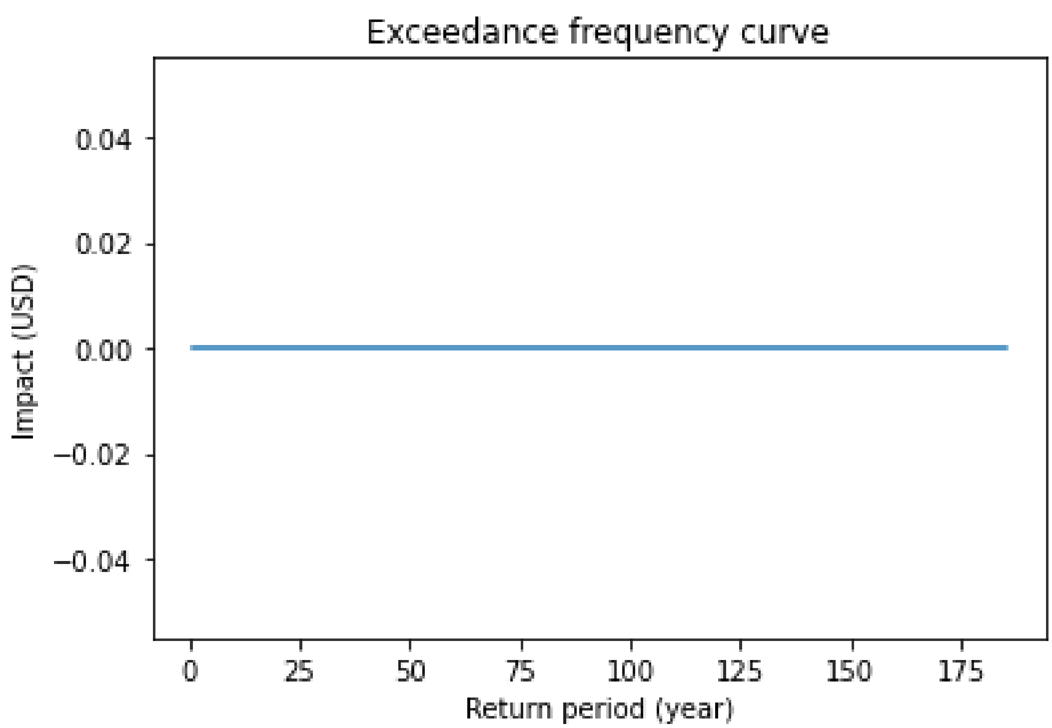


1e10
2.6
2.4
2.2
2.0
1.8
1.6
1.4
Value (USD)
27.5°N
27°N
26.5°N
26°N
25.5°N
81.5°W
81°W
80.5°W
80°W
79.5°W
Miami


TC max intensity at each point
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
Intensity (m/s)
70
60
50
40
30
20


TC max intensity at each point
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
70
60
50
40
30
20
10
0
Intensity (m/s)





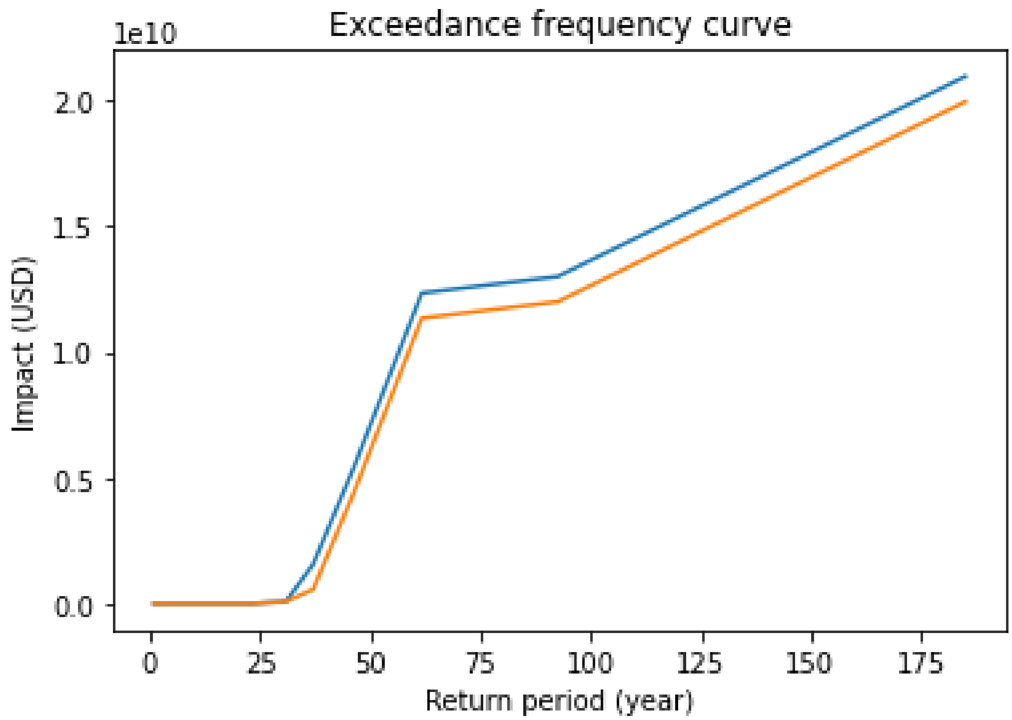

```
# effect of risk_transf_attach and risk_transf_cover
import numpy as np
from climada.entity import ImpactFuncSet, ImpfTropCyclone, Exposures
from climada.entity.measures import Measure
from climada.hazard import Hazard
from climada.engine import ImpactCalc
```

(continues on next page)

(continued from previous page)

```
from climada.util import HAZ_DEMO_H5, EXP_DEMO_H5

# define measure
meas = Measure(
    name='Insurance',
    haz_type='TC',
    color_rgb=np.array([1, 1, 1]),
    cost=500000000,
    risk_transf_attach=5.0e8,
    risk_transf_cover=1.0e9,
)

# impact functions
impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_all = ImpactFuncSet([impf_tc])

# Hazard
haz = Hazard.from_hdf5(HAZ_DEMO_H5)
haz.check()

# Exposures
exp = Exposures.from_hdf5(EXP_DEMO_H5)
exp.check()

# impact before
imp = ImpactCalc(exp, impf_all, haz).impact()
ax = imp.calc_freq_curve().plot(label='original');

# impact after. risk_transf will be added to the cost of the measure
imp_new, risk_transf = meas.calc_impact(exp, impf_all, haz)
imp_new.calc_freq_curve().plot(axis=ax, label='measure');
print('risk_transfer {:.3}'.format(risk_transf.aai_agg))
```

```
2022-03-30 20:10:29,899 - climada.hazard.base - INFO - Reading /home/yuyue/climada/
↪demo/data/tc_fl_1990_2004.h5
2022-03-30 20:10:30,001 - climada.entity.exposures.base - INFO - Reading /home/yuyue/
↪climada/demo/data/exp_demo_today.h5
2022-03-30 20:10:30,030 - climada.entity.exposures.base - INFO - centr_ not set.
2022-03-30 20:10:30,034 - climada.entity.exposures.base - INFO - Matching 50␣
↪exposures with 2500 centroids.
2022-03-30 20:10:30,035 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-03-30 20:10:30,047 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 216 events.
2022-03-30 20:10:30,084 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-30 20:10:30,087 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 216 events.
risk_transfer 2.7e+07
```



10.3.2 MeasureSet class

Similarly to the `ImpactFuncSet`, `MeasureSet` is a container which handles `Measure` instances through the methods `append()`, `extend()`, `remove_measure()` and `get_measure()`. Use the `check()` method to make sure all the measures have been properly set.

For a complete class documentation, refer to the Python modules docs: *climada.entity.measures.measure_set.MeasureSet*

```
# build measures
import numpy as np
import matplotlib.pyplot as plt
from climada.entity.measures import Measure, MeasureSet

meas_1 = Measure(
    haz_type='TC',
    name='Mangrove',
    color_rgb=np.array([1, 1, 1]),
    cost=500000000,
    mdd_impact=(1, 2),
    paa_impact=(1, 2),
    hazard_inten_imp=(1, 2),
    risk_transf_cover=500,
)

meas_2 = Measure(
    haz_type='TC',
    name='Sandbags',
    color_rgb=np.array([1, 1, 1]),
    cost=22000000,
```

(continues on next page)

(continued from previous page)

```
    mdd_impact=(1, 2),
    paa_impact=(1, 3),
    hazard_inten_imp=(1, 2),
    exp_region_id=2,
)

# gather all measures
meas_set = MeasureSet()
meas_set.append(meas_1)
meas_set.append(meas_2)
meas_set.check()

# select one measure
meas_sel = meas_set.get_measure(name='Sandbags')
print(meas_sel[0].name, meas_sel[0].cost)
```

```
Sandbags 22000000
```

10.3.3 Read measures of an Excel file

Measures defined in an excel file following the template provided in sheet `measures` of `climada_python/data/system/entity_template.xlsx` can be ingested directly using the method `from_excel()`.

```
from climada.entity.measures import MeasureSet
from climada.util import ENT_TEMPLATE_XLS

# Fill DataFrame from Excel file
file_name = ENT_TEMPLATE_XLS # provide absolute path of the excel file
meas_set = MeasureSet.from_excel(file_name)
meas_set
```

```
<climada.entity.measures.measure_set.MeasureSet at 0x1f30d913b80>
```

10.3.4 Write measures

Measures can be writen in Excel format using `write_excel()` method.

```
from climada.entity.measures import MeasureSet
from climada.util import ENT_TEMPLATE_XLS

# Fill DataFrame from Excel file
file_name = ENT_TEMPLATE_XLS # provide absolute path of the excel file
meas_set = MeasureSet.from_excel(file_name)

# write file
meas_set.write_excel('results/tutorial_meas_set.xlsx')
```

Pickle can always be used as well:

```
from climada.util.save import save
# this generates a results folder in the current path and stores the output there
save('tutorial_meas_set.p', meas_set)
```

10.4 DiscRates class

Discount rates are used to calculate the net present value of any future or past value. They are thus used to compare amounts paid (costs) and received (benefits) in different years. A project is economically viable (attractive), if the net present value of benefits exceeds the net present value of costs - a cost-benefit ratio < 1.

There are several important implications that come along with discount rates. Namely, that higher discount rates lead to smaller net present values of future impacts (costs). As a consequence of that, climate action and mitigation measures can be postboned. In the literature higher discount rates are typically justified by the expectation of continued exponential growth of the economy. The most widley used interest rate in climate change economics is 1.4% as propsed by the Stern Review (2006). Neoliberal economists around Nordhaus (2007) claim that rates should be higher, around 4.3%. Environmental economists argue that future costs shouldn't be discounted at all. This is especially true for non-monetary variables such as ecosystems or human lifes, where no price tag should be applied out of ethical reasons. This discussion has a long history, reaching back to the 18th century: "Some things have a price, or relative worth, while other things have a dignity, or inner worth" (Kant, 1785).

This class contains the discount rates for every year and discounts given values. Its attributes are:

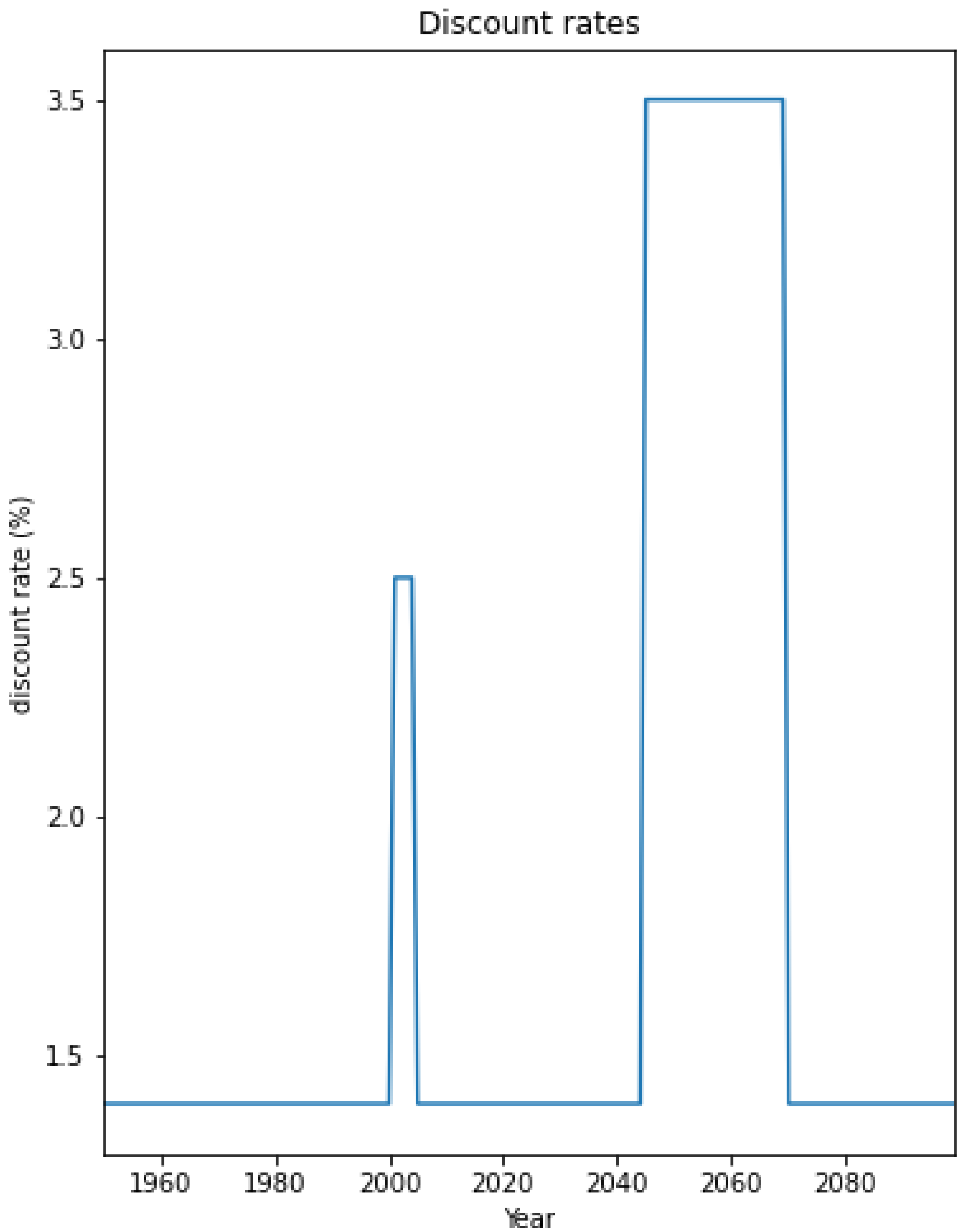
- years (np.array): years
- rates (np.array): discount rates for each year (between 0 and 1)

For a complete class documentation, refer to the Python modules docs: *climada.entity.disc_rates.base.DiscRates*

An example of use - we define discount rates and apply them on a coastal protection scheme which initially costs 100 mn. USD plus 75'000 USD mainteance each year, starting after 10 years. Net present value of the project can be calculated as displayed:

```
%matplotlib inline
import numpy as np
from climada.entity import DiscRates

# define discount rates
years = np.arange(1950, 2100)
rates = np.ones(years.size) * 0.014
rates[51:55] = 0.025
rates[95:120] = 0.035
disc = DiscRates(years=years, rates=rates)
disc.plot()

# Compute net present value between present year and future year.
ini_year = 2019
end_year = 2050
val_years = np.zeros(end_year-ini_year+1)
val_years[0] = 100000000 # initial investment
val_years[10:] = 75000 # maintenance from 10th year
npv = disc.net_present_value(ini_year, end_year, val_years)
print('net present value: {:.5e}'.format(npv))
```

```
net present value: 1.01231e+08
```



Read discount rates of an Excel file

Discount rates defined in an excel file following the template provided in sheet `discount` of `climada_python/climada/data/system/entity_template.xlsx` can be ingested directly using the method `from_excel()`.

```
from climada.entity import DiscRates
from climada.util import ENT_TEMPLATE_XLS

# Fill DataFrame from Excel file
file_name = ENT_TEMPLATE_XLS # provide absolute path of the excel file
print('Read file:', ENT_TEMPLATE_XLS)
```

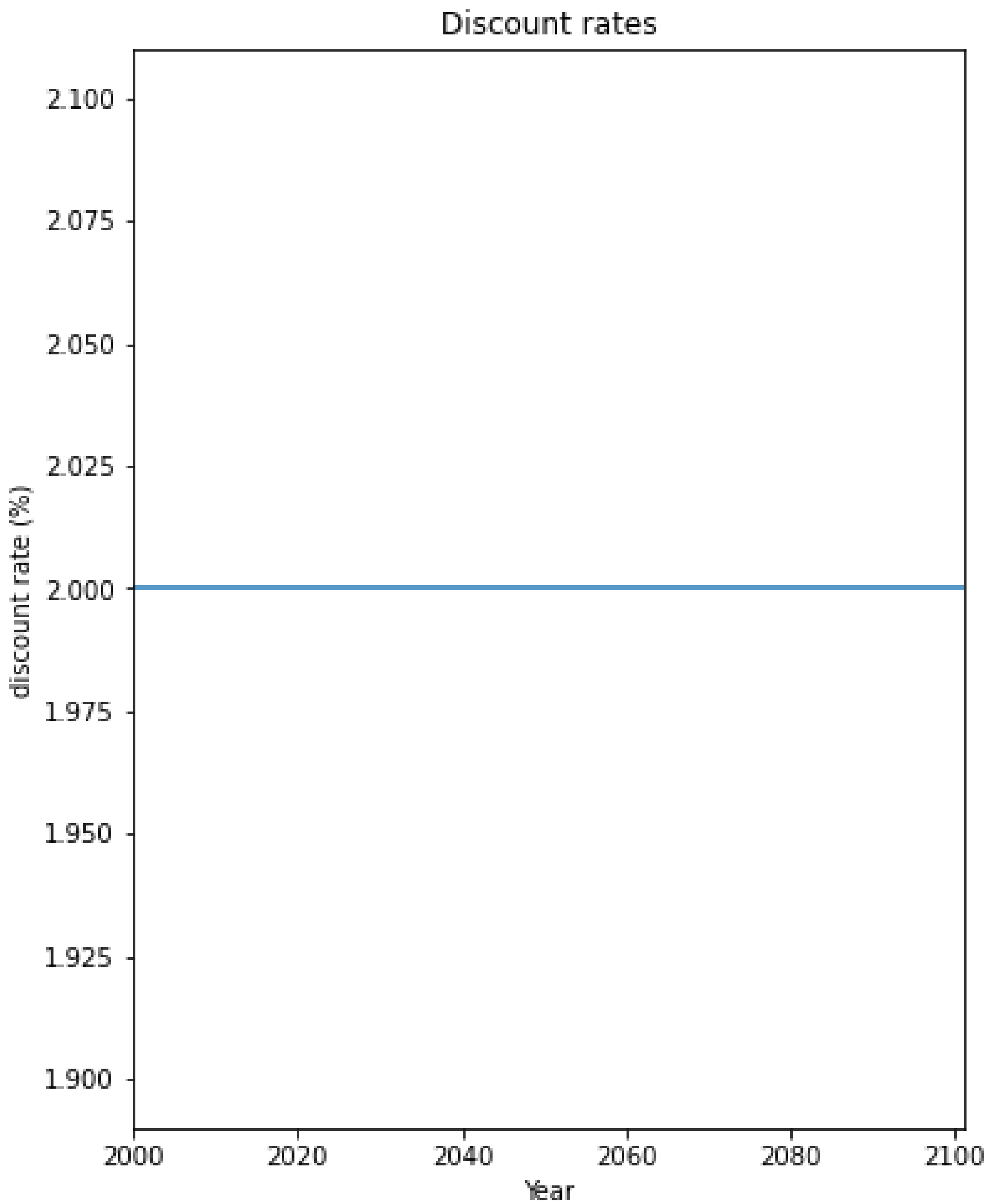
(continues on next page)

(continued from previous page)

```
disc = DiscRates.from_excel(file_name)
disc.plot();
```

```
Read file: /Users/ckropf/climada/data/entity_template.xlsx
```

```
<AxesSubplot:title={'center':'Discount rates'}, xlabel='Year', ylabel='discount rate (
↪%)'>
```



10.4.1 Write discount rates

Users may write the discounts in Excel format using write_excel() method `write_excel()`.

```
from climada.entity import DiscRates
from climada.util import ENT_TEMPLATE_XLS

# Fill DataFrame from Excel file
file_name = ENT_TEMPLATE_XLS # provide absolute path of the excel file
disc = DiscRates.from_excel(file_name)

# write file
disc.write_excel('results/tutorial_disc.xlsx')
```

Pickle can always be used as well:

```
from climada.util.save import save
# this generates a results folder in the current path and stores the output there
save('tutorial_disc.p', disc)
```

10.5 Impact Data functionalities

Import data from EM-DAT CSV file and populate Impact()-object with the data.

The core functionality of the module is to read disaster impact data as downloaded from the International Disaster Database EM-DAT (www.emdat.be) and produce a CLIMADA Impact()-instance from it. The purpose is to make impact data easily available for comparison with simulated impact inside CLIMADA, e.g. for calibration purposes.

10.5.1 Data Source

The International Disaster Database EM-DAT www.emdat.be

Download: https://public.emdat.be/ (register for free and download data to continue)

10.5.2 Most important functions

- clean_emdat_df: read CSV from EM-DAT into a DataFrame and clean up.
- emdat_to_impact: create Impact-instance populated with impact data from EM-DAT data (CSV).
- emdat_countries_by_hazard: get list of countries affected by a certain haazrd (disaster (sub-)type) in EM-DAT.
- emdat_impact_yearlysum: create DataFrame with impact from EM-DAT summed per country and year.

10.5.3 Demo data

The demo data used here (demo_emdat_impact_data_2020.csv) contains entries for the disaster subtype "Tropical cyclone" from 2000 to 2020.

```
"""Load required packages and set path to CSV-file from EM-DAT"""

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

from climada.util.constants import DEMO_DIR
from climada.engine.impact_data import emdat_countries_by_hazard, \
    emdat_impact_yearlysum, emdat_to_impact, clean_emdat_df

# set path to CSV file downloaded from https://public.emdat.be :
emdat_file_path = DEMO_DIR.joinpath('demo_emdat_impact_data_2020.csv')
```

clean_emdat_df()

read CSV from EM-DAT into a DataFrame and clean up.

Use the parameters countries, hazard, and year_range to filter. These parameters are the same for most functions shown here.

```
"""Create DataFrame df with EM-DAT entries of tropical cyclones in Thailand and Viet␣
↪Nam in the years 2005 and 2006"""

df = clean_emdat_df(emdat_file_path, countries=['THA', 'Viet Nam'], hazard=['TC'], \
                    year_range=[2005, 2006])
print(df)
```

```
          Dis No  Year  Seq Disaster Group Disaster Subgroup Disaster Type  \
0  2005-0540-VNM  2005  540        Natural    Meteorological         Storm
1  2005-0540-THA  2005  540        Natural    Meteorological         Storm
2  2005-0536-VNM  2005  536        Natural    Meteorological         Storm
3  2005-0611-VNM  2005  611        Natural    Meteorological         Storm
4  2006-0362-VNM  2006  362        Natural    Meteorological         Storm
5  2006-0648-VNM  2006  648        Natural    Meteorological         Storm
6  2006-0251-VNM  2006  251        Natural    Meteorological         Storm
7  2006-0517-VNM  2006  517        Natural    Meteorological         Storm

   Disaster Subtype  Disaster Subsubtype         Event Name Entry Criteria  \
0  Tropical cyclone                  NaN             Damrey           Kill
1  Tropical cyclone                  NaN             Damrey           Kill
2  Tropical cyclone                  NaN            Vicente           Kill
3  Tropical cyclone                  NaN       Kai Tak (21)           Kill
4  Tropical cyclone                  NaN              Bilis           Kill
5  Tropical cyclone                  NaN     Durian (Reming)          Kill
6  Tropical cyclone                  NaN     Chanchu (Caloy)          Kill
7  Tropical cyclone                  NaN  Xangsane (Milenyo)          Kill

   ... End Day Total Deaths No Injured No Affected No Homeless Total Affected  \
0  ...    30.0         75.0        28.0    337632.0         NaN       337660.0
1  ...    30.0         10.0         NaN      2000.0         NaN         2000.0
2  ...    19.0          8.0         NaN      8500.0         NaN         8500.0
```

(continues on next page)

(continued from previous page)

```
3  ...     4.0         20.0        NaN     15000.0         NaN        15000.0
4  ...    19.0         17.0        NaN         NaN      2000.0         2000.0
5  ...     8.0         95.0     1360.0    975000.0    250000.0      1226360.0
6  ...    17.0        204.0        NaN    600000.0         NaN       600000.0
7  ...     6.0         71.0      525.0   1368720.0     98680.0      1467925.0

  Reconstruction Costs ('000 US$) Insured Damages ('000 US$)  \
0                             NaN                        NaN
1                             NaN                        NaN
2                             NaN                        NaN
3                             NaN                        NaN
4                             NaN                        NaN
5                             NaN                        NaN
6                             NaN                        NaN
7                             NaN                        NaN

  Total Damages ('000 US$)        CPI
0                 219250.0  76.388027
1                  20000.0  76.388027
2                  20000.0  76.388027
3                  11000.0  76.388027
4                      NaN  78.852256
5                 456000.0  78.852256
6                      NaN  78.852256
7                 624000.0  78.852256

[8 rows x 43 columns]
```

emdat_countries_by_hazard()

Pick a hazard and a year range to get a list of countries affected from the EM-DAT data.

```
"""emdat_countries_by_hazard: get lists of countries impacted by tropical cyclones␣
↪from 2010 to 2019"""

iso3_codes, country_names = emdat_countries_by_hazard(emdat_file_path, hazard='TC',␣
↪year_range=(2010, 2019))

print(country_names)

print(iso3_codes)
```

```
['China', 'Dominican Republic', 'Antigua and Barbuda', 'Fiji', 'Australia',
↪'Bangladesh', 'Belize', 'Barbados', 'Cook Islands', 'Canada', 'Bahamas', 'Guatemala
↪', 'Jamaica', 'Saint Lucia', 'Madagascar', 'Mexico', "Korea, Democratic People's␣
↪Republic of", 'El Salvador', 'Myanmar', 'French Polynesia', 'Solomon Islands',
↪'Taiwan, Province of China', 'India', 'United States of America', 'Honduras', 'Haiti
↪', 'Pakistan', 'Philippines', 'Hong Kong', 'Korea, Republic of', 'Nicaragua', 'Oman
↪', 'Japan', 'Puerto Rico', 'Thailand', 'Martinique', 'Papua New Guinea', 'Tonga',
↪'Venezuela, Bolivarian Republic of', 'Viet Nam', 'Saint Vincent and the Grenadines',
↪ 'Vanuatu', 'Dominica', 'Cuba', 'Comoros', 'Mozambique', 'Malawi', 'Samoa', 'South␣
↪Africa', 'Sri Lanka', 'Palau', 'Wallis and Futuna', 'Somalia', 'Seychelles',
↪'Réunion', 'Kiribati', 'Cabo Verde', 'Micronesia, Federated States of', 'Panama',
↪'Costa Rica', 'Yemen', 'Tuvalu', 'Northern Mariana Islands', 'Colombia', 'Anguilla',
↪ 'Djibouti', 'Cambodia', 'Macao', 'Indonesia', 'Guadeloupe', 'Turks and Caicos␣
```

(continues on next page)

(continued from previous page)

```
↪Islands', 'Saint Kitts and Nevis', "Lao People's Democratic Republic", 'Mauritius',
↪'Marshall Islands', 'Portugal', 'Virgin Islands, U.S.', 'Zimbabwe', 'Saint␣
↪Barthélemy', 'Virgin Islands, British', 'Saint Martin (French part)', 'Sint Maarten␣
↪(Dutch part)', 'Tanzania, United Republic of']
['CHN', 'DOM', 'ATG', 'FJI', 'AUS', 'BGD', 'BLZ', 'BRB', 'COK', 'CAN', 'BHS', 'GTM',
↪'JAM', 'LCA', 'MDG', 'MEX', 'PRK', 'SLV', 'MMR', 'PYF', 'SLB', 'TWN', 'IND', 'USA',
↪'HND', 'HTI', 'PAK', 'PHL', 'HKG', 'KOR', 'NIC', 'OMN', 'JPN', 'PRI', 'THA', 'MTQ',
↪'PNG', 'TON', 'VEN', 'VNM', 'VCT', 'VUT', 'DMA', 'CUB', 'COM', 'MOZ', 'MWI', 'WSM',
↪'ZAF', 'LKA', 'PLW', 'WLF', 'SOM', 'SYC', 'REU', 'KIR', 'CPV', 'FSM', 'PAN', 'CRI',
↪'YEM', 'TUV', 'MNP', 'COL', 'AIA', 'DJI', 'KHM', 'MAC', 'IDN', 'GLP', 'TCA', 'KNA',
↪'LAO', 'MUS', 'MHL', 'PRT', 'VIR', 'ZWE', 'BLM', 'VGB', 'MAF', 'SXM', 'TZA']
```

emdat_to_impact()

function to load EM-DAT impact data and return impact set with impact per event

Parameters:

- emdat_file_csv (str): Full path to EMDAT-file (CSV)
- hazard_type_climada (str): Hazard type abbreviation used in CLIMADA, e.g. 'TC'

Optional parameters:

- hazard_type_emdat (list or str): List of Disaster (sub-)type according EMDAT terminology or CLIMADA hazard type abbreviations. e.g. ['Wildfire', 'Forest fire'] or ['BF']
- year_range (list with 2 integers): start and end year e.g. [1980, 2017]
- countries (list of str): country ISO3-codes or names, e.g. ['JAM', 'CUB']. Set to None or ['all'] for all countries
- reference_year (int): reference year of exposures for normalization. Impact is scaled proportional to GDP to the value of the reference year. No scaling for reference_year=0 (default)
- imp_str (str): Column name of impact metric in EMDAT CSV, e.g. 'Total Affected'; default = "Total Damages"

Returns:

- impact_instance (instance of climada.engine.Impact): Impact() instance (same format as output from CLIMADA impact computations). Values are scaled with GDP to reference_year if reference_year not equal 0. impact_instance.eai_exp holds expected annual impact for each country. impact_instance.coord_exp holds rough central coordinates for each country.
- countries (list): ISO3-codes of countries imn same order as in impact_instance.eai_exp

```
"""Global TC damages 2000 to 2009"""

impact_emdat, countries = emdat_to_impact(emdat_file_path, 'TC', year_range=(2000,
↪2009))

print('Number of TC events in EM-DAT 2000 to 2009 globally: %i' %(impact_emdat.event_
↪id.size))
```

(continues on next page)

(continued from previous page)

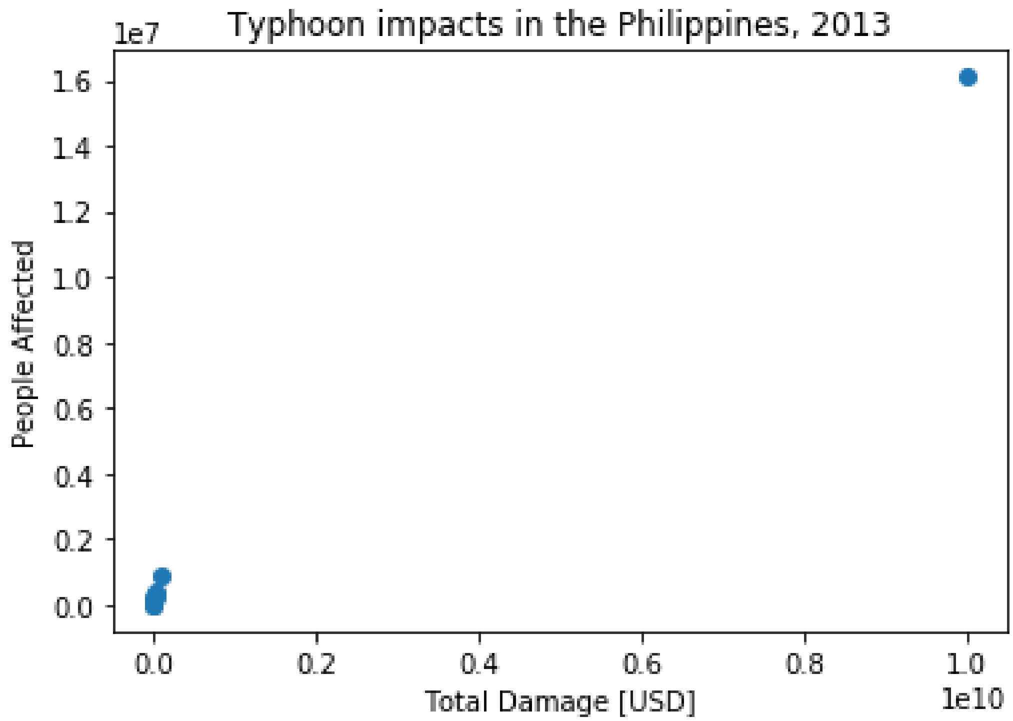
```
print('Global annual average monetary damage (AAI) from TCs as reported in EM-DAT␣
↪2000 to 2009: USD billion %2.2f' \
      %(impact_emdat.aai_agg/1e9))
```

```
2021-10-19 16:44:58,210 - climada.engine.impact_data - WARNING - ISO3alpha code not␣
↪found in iso_country: SPI
2021-10-19 16:44:59,007 - climada.engine.impact_data - WARNING - Country not found in␣
↪iso_country: SPI
Number of TC events in EM-DAT 2000 to 2009 globally: 533
Global annual average monetary damage (AAI) from TCs as reported in EM-DAT 2000 to␣
↪2009: USD billion 38.07
```

```
"""Total people affected by TCs in the Philippines in 2013:"""

# People affected
impact_emdat_PHL, countries = emdat_to_impact(emdat_file_path, 'TC', countries='PHL',␣
↪\
                                    year_range=(2013,2013), imp_str="Total Affected")

print('Number of TC events in EM-DAT in the Philipppines, 2013: %i' \
      %(impact_emdat_PHL.event_id.size))
print('\nPeople affected by TC events in the Philippines in 2013 (per event):')
print(impact_emdat_PHL.at_event)
print('\nPeople affected by TC events in the Philippines in 2013 (total):')
print(int(impact_emdat_PHL.aai_agg))

# Comparison to monetary damages:
impact_emdat_PHL_USD, _ = emdat_to_impact(emdat_file_path, 'TC', countries='PHL', \
                                    year_range=(2013,2013))

ax = plt.scatter(impact_emdat_PHL_USD.at_event, impact_emdat_PHL.at_event)
plt.title('Typhoon impacts in the Philippines, 2013')
plt.xlabel('Total Damage [USD]')
plt.ylabel('People Affected');
#plt.xscale('log')
#plt.yscale('log')
```

```
Number of TC events in EM-DAT in the Philipppines, 2013: 8

People affected by TC events in the Philippines in 2013 (per event):
[7.269600e+04 1.059700e+04 8.717550e+05 2.204430e+05 1.610687e+07
 3.596000e+03 3.957300e+05 2.628840e+05]

People affected by TC events in the Philippines in 2013 (total):
17944571
```

```
Text(0, 0.5, 'People Affected')
```



emdat_impact_yearlysum()

function to load EM-DAT impact data and return DataFrame with impact summed per year and country

Parameters:

- emdat_file_csv (str): Full path to EMDAT-file (CSV)

Optional parameters:

- hazard (list or str): List of Disaster (sub-)type according EMDAT terminology or CLIMADA hazard type abbreviations. e.g. ['Wildfire', 'Forest fire'] or ['BF']
- year_range (list with 2 integers): start and end year e.g. [1980, 2017]
- countries (list of str): country ISO3-codes or names, e.g. ['JAM', 'CUB']. Set to None or ['all'] for all countries
- reference_year (int): reference year of exposures for normalization. Impact is scaled proportional to GDP to the value of the reference year. No scaling for reference_year=0 (default)
- imp_str (str): Column name of impact metric in EMDAT CSV, e.g. 'Total Affected'; default = "Total Damages"
- version (int): given EM-DAT data format version (i.e. year of download), changes naming of columns/variables (default: 2020)

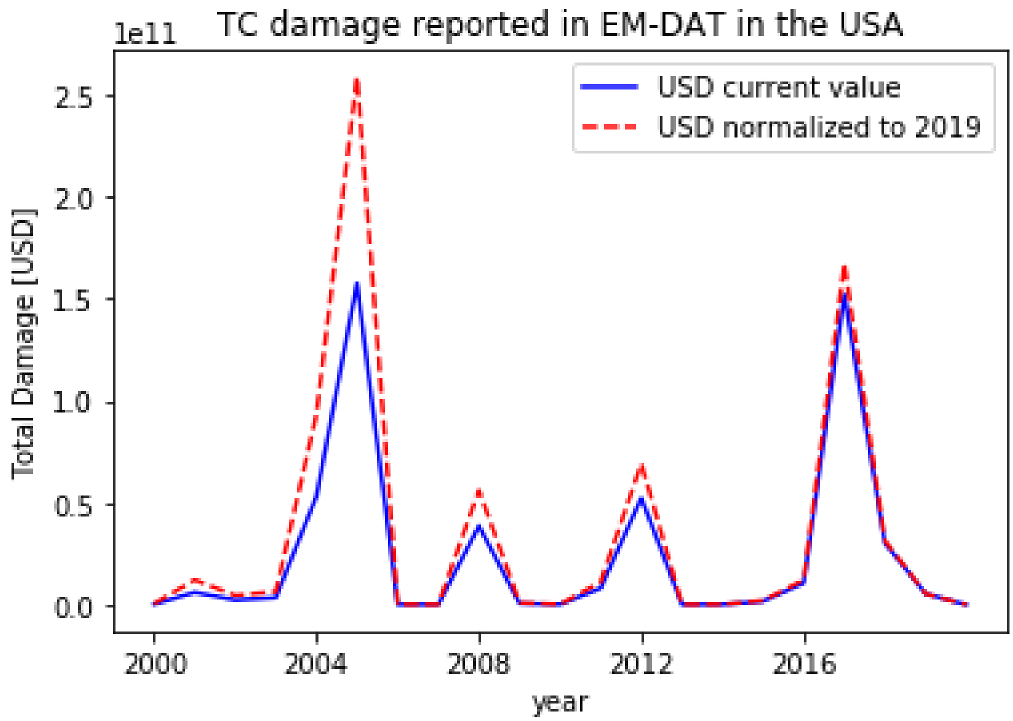
Returns:

- pandas.DataFrame with impact per year and country

```
"""Yearly TC damages in the USA, normalized and current"""

yearly_damage_normalized_to_2019 = emdat_impact_yearlysum(emdat_file_path, countries=
↪'USA', \
                                                          hazard='Tropical cyclone',␣
↪year_range=None, \
                                                          reference_year=2019)

yearly_damage_current = emdat_impact_yearlysum(emdat_file_path, countries=['USA'],␣
↪hazard='TC',)

import matplotlib.pyplot as plt

fig, axis = plt.subplots(1, 1)
axis.plot(yearly_damage_current.year, yearly_damage_current.impact, 'b', label='USD␣
↪current value')
axis.plot(yearly_damage_normalized_to_2019.year, yearly_damage_normalized_to_2019.
↪impact_scaled, \
          'r--', label='USD normalized to 2019')
plt.legend()
axis.set_title('TC damage reported in EM-DAT in the USA')
axis.set_xticks([2000, 2004, 2008, 2012, 2016])
axis.set_xlabel('year')
axis.set_ylabel('Total Damage [USD]');
```

```
[2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2014
 2015 2016 2017 2018 2019 2020]
```

```
Text(0, 0.5, 'Total Damage [USD]')
```



10.6 END-TO-END COST BENEFIT CALCULATION

10.6.1 Contents

- *Introduction*
- *What is a cost-benefit?*
- *CostBenefit class data structure*
- *Detailed CostBenefit calculation: LitPop + TropCyclone*
- *Conclusion*

10.6.2 Introduction

The goal of this tutorial is to show a full end-to-end cost-benefit calculation. Note that this tutorial shows the work flow and some data exploration, but does not explore all possible features.

The tutorial will start with an explanation of the mathematics of an cost-benefit calculation, and then will move on to how this is implemented in CLIMADA. We will then go through a few end-to-end calculations.

If you just need to see the code in action, you can skip these first parts, which are mostly for reference.

The tutorial assumes that you're already familiar with CLIMADA's ***Hazard***, ***Exposures***, ***impact functions***, ***Impact*** and ***adaptation measure*** functionality. The cost-benefit calculation is often the last part of an analyses, and it brings all the previous components together.

10.6.3 What is a cost-benefit?

A cost-benefit analysis in CLIMADA lets you compare the effectiveness of different hazard adaptation options.

The cost-benefit ratio describes how much loss you can prevent per dollar of expenditure (or whatever currency you're using) over a period of time. When a cost-benefit ratio is less than 1, the cost is less than the benefit and CLIMADA is predicting a worthwhile investment. Smaller ratios therefore represent better investments. When a cost-benefit is greater than 1, the cost is more than the benefit and the offset losses are less than the cost of the adaptation measure: based on the financials alone, the measure may not be worth it. Of course, users may have factors beyond just cost-benefits that influence decisions.

CLIMADA doesn't limit cost-benefits to just financial exposures. The cost-benefit ratio could represent hospitalisations avoided per Euro spent, or additional tons of crop yield per Swiss Franc.

The cost-benefit calculation has a few complicated components, so in this section we'll build up the calculation step by step.

Simple cost-benefits

The simplest form of a cost-benefit calculation goes like this:

$$\text{CostBenefit} = \frac{\text{cost}}{\text{benefit}} = \frac{\text{cost}}{N \cdot (\text{AAI without measures} - \text{AAI with measures})}$$

where cost is the cost of implementing a set of measures, the AAI is the average annual impact from your hazard event set on your exposure, and N is the number of years the cost-benefit is being evaluated over.

Note that:

- Whether an adaptation measure is seen to be effective might depend on the number of years you are evaluating the cost-benefit over. For example, a €50 mn investment that prevents an average of €1 mn losses per year will only 'break even' after $N = 50$ years.
- Since an adaptation measure could in theory make an impact worse (a negative benefit) it is possible to have negative cost-benefit ratios.
- CLIMADA allows you to use other statistics than annual average impact, but to keep thing simple we'll use average annual impact throughout this tutorial.

Time-dependence

The above equation works well when the only thing changing is an adaptation measure. But usually CLIMADA cost-benefit calculation will want to describe a climate and exposure that also change over time. In this case it's not enough to multiply the change in average annual impact by the number of years we're evaluating over, and we need to calculate a benefit for every year separately and sum them up.

We can modify the benefit part of cost-benefit to reflect this. CLIMADA doesn't assume that the user will have explicit hazard and impact objects for every year in the study period, and so interpolates between the impacts at the start and the end of the period of interest. If we're evaluating between years T_0, usually close to the present, and T_1 in the future, then we can say:

$$\text{benefit} = \sum_{t=T_0}^{T_1} \alpha(t) \bigg(\text{AAI with measures}_{T_1} - \text{AAI with measures}_{T_0} \bigg) - N * \text{AAI without measure}_{T_0}$$

Where $\alpha(t)$ is a function of the year t describing the interpolation of hazard and exposure values between T_0 and T_1. The function returns values in the range $[0, 1]$, usually with $\alpha(T_0) = 0$ and $\alpha(T_0) = 1$.

Note that:

- This calculation now requires three separate impact calculations: present-day impacts without measures implemented, present-day impacts with measures implemented, and future impacts with measures implemented.
- Setting $\alpha(t) = 1$ for all values of t simplifies this to the first cost-benefit equation above.

CLIMADA lets you set $\alpha(t)$ to 1 for all years t, or as \$$\alpha_k(t) = \frac{(t-T_0)^k}{(T_1-T_0)^k}$ for $t \in [T_0, T_1]$\$

where k is user-provided, called `imp_time_depen`. This expression is a polynomial curve between T_0 and T_1 normalised so that $\alpha_k(T_0) = 0$ and $\alpha_k(T_1) = 1$. The choice of k determines how quickly the transition occurs between the present and future. When $k = 1$ the function is a straight line. When $k > 1$ change begins slowly and speeds up over time. When $k < 1$ change is begins quickly and slows over time.

If this math is tough, the key takeaways are

- Cost benefit calculations take a long view, summing the benefits of adaptation measures over many years in a changing world
- CLIMADA describes how the change from the present to the future scenarios happens with the `imp_time_depen` parameter. With values < 1 the change starts quickly and slows down. When it is equal to 1 change is steady. When it's > 1 change starts slowly and speeds up.

Discount rates

The final addition to our cost-benefit calculation is the *discount rate*.

The discount rate tries to formalise an idea from economics that says that a gain in the future is worth less to us than the same gain right now. For example, paying €1 mn to offset €2 mn of economic losses next year is 'worth more' than paying €1 mn to offset €2 mn of economic losses in 2080.

In practice it provides a way to future monetary values to an estimated worth today, called their *net present value*. Note that this is *not* an adjustment for inflation.

The choice of discount rate is a contentious topic in adaptation finance, since it can strongly affect a cost-benefit calculation. The most widley used discount rate in climate change economics is 1.4% as proposed by the Stern Review (2006). Neoliberal economists around Nordhaus (2007) claim that rates should be higher, around 4.3%, reflecting continued economic growth and a society that will be better at adapting in the future compared to now. Environmental economists argue that future costs shouldn't be discounted at all.

To illustrate, with a 1.4% annual discount rate, a gain of €100 next year is equivalent to €98.60 this year, and a gain of €100 in 15 years is equivalent to €$(100 * 0.986^{15})$ = € 80.94 this year. With a rate of 4.3% this drops to €51.72.

We can add this into the cost-benefit calculation by defining $d(t)$, the discount rate for each year t. A constant rate of 1.4% would then set $d(t) = 0.014$ for all values of t.

Then the adjustment $D(t)$ from year t to the net present value in year T_0 is given by

$$D(t) = \prod_{y=T_0}^{t} (1 - d(y))$$

With a constant 1.4% discount rate, we have $D(t) = 0.986^{t-T_0}$. With a discount rate of zero we have $D(t) = 1$.

Adding this to our equation for total benefits we get:

$$\text{benefit} = \sum_{t=T_0}^{T_1} \alpha(t) D(t) (\text{AAI with measures}_{T_1} - \text{AAI with measures}_{T_0}) - N * \text{AAI without measure}_{T_0}$$

Note:

- Setting the rates to zero ($d(t) = 0$) means $D(t) = 1$ and the term drops out of the equation.

- Be careful with your choice of discount rate when your exposure is non-economic. It can be hard to justify applying rates to e.g. ecosystems or human lives.

10.6.4 CostBenefit class data structure

The CostBenefit class does not require any attributes to be defined by the user. All attributes are set from parameters when the method `CostBenefit.calc()` is called.

After calling the `calc` method the `CostBenefit` object has the following attributes:

Attributes created in `CostBenefit.calc`	Data Type	Description
present_year	int	The current year
future_year	int	The future scenario year
tot_climate_risk	float	The total climate risk in the present scenario, evaluated according to the provided risk function (annual average impact by default)
unit	string	Units to measure impact
benefit	dict(flc	The benefit of each measure, keyed by measure name
cost_ben_ratio	dict(flc	The cost benefit of each measure, keyed by measure name
imp_meas_future	dict(di	Dictionaries describing the impacts of each measure in the future scenario. Keyed by measure name (with 'no measure' for no measures). The entries in each dictionary are described below.
imp_meas_present	dict(di	Dictionaries describing the impacts of each measure in the present-day scenario. Keyed by measure name (with 'no measure' for no measures). The entries in each dictionary are described below.

Each dictionary stored in the attributes `imp_meas_future` and `imp_meas_present` has entries:

Key	Data Type	Description
cost	tuple (cost measure, cost factor insurance)	The cost of implementing the measure, and the cost factor if risk transfers are being calculated
impact	Impact	Impact object calculated with the present (`imp_meas_present`) or future (`imp_meas_future`) hazard, exposure and impact functions
risk	float	A value of annual risk used in the cost-benefit calculation. A summary statistic calculated from the Impact object. Most commonly the average annual impact, but can be changed with the `CostBenefit.calc`'s `risk_func`parameter.
risk_t	float	Annual expected risk transfer (if calculated)
efc	ImpactFreqCurve	The impact exceedance freq for this measure calculated from the Impact object (if calculated)

The dictionary will also include a 'no measure' entry with the same structure, giving the impact analysis when no measures are implemented.

The `calc` calculation

Let's look at the parameters needed for the calculation:

```
CostBenefit.calc(hazard, entity, haz_future=None, ent_future=None,
                 future_year=None, risk_func=risk_aai_agg, imp_time_depen=None, save_
↪imp=False)
```

These are:

- `hazard` (Hazard object): the present-day or baseline hazard event set
- `entity` (Entity object): the present-day or baseline Entity object. `Entity` is the container class containing
  - `exposure` (Exposures object): the present-day or baseline exposure
  - `disc_rates` (DiscRates object): the discount rates to be applied in the cost-benefit calculation. Only discount rates from `entity` and not `ent_future` are used.
  - `impact_funcs` (ImpactFuncSet object): the impact functions required to calculate impacts from the present-day hazards and exposures
  - `measures` (MeasureSet object): the set of measures to implement in the analysis. This will almost always be the same as the measures in the `ent_future` Entity (if set).
- `haz_future` (Hazard object, optional): the future hazard event set, if different from present.
- `ent_future` (Entity object, optional): the future Entity, if different from present. Note that the same adaptation measures must be present in both `entity` and `ent_future`.
- `future_year` (int): the year of the future scenario. This is only used if the Entity's `exposures.ref_year` isn't set, or no future entity is provided.
- `risk_func` (function): this is the risk function used to describe the annual impacts used to describe benefits. The default is `risk_aai_agg`, the average annual impact on the Exposures (defined in the CostBenefit module). This function can be replaces with any function that takes an Impact object as input and returns a number. The CostBenefit module provides two others functions `risk_rp_100` and `risk_rp_250`, the 100-year and 250-year return period impacts respectively.
- `imp_time_depen` (float): This describes how hazard and exposure evolve over time in the calculation. In the descriptions above this is the parameter k defining $\alpha_k(t)$. When > 1 change is superlinear and occurs nearer the start of the analysis. When < 1 change is sublinear and occurs nearer the end.
- `save_imp` (boolean): whether to save the hazard- and location-specific impact data. This is used in a lot of follow-on calculations, but is very large if you don't need it.

10.6.5 Detailed CostBenefit calculation: LitPop + TropCyclone

We present a detailed example for the hazard ***Tropical Cyclones*** and the exposures from ***LitPop*** .

To speed things up we'll use the CLIMADA Data API to download the data as needed. The data download roughly follows the ***Data API tutorial*** for Haiti. If this is a problem you can build tropical cyclone event sets, and LitPop exposures following the relevant tutorials.

Download hazard

We will get data for present day tropical cyclone hazard in Haiti, and for 2080 hazard under the RCP 6.0 warming scenario. Note that the Data API provides us with a full event set of wind footprints rather than a TCTracks track dataset, meaning we don't have to generate the wind fields ourselves.

```
from climada.util.api_client import Client

client = Client()
future_year = 2080
haz_present = client.get_hazard('tropical_cyclone',
                                properties={'country_name': 'Haiti',
                                            'climate_scenario': 'historical',
                                            'nb_synth_tracks':'10'})
haz_future = client.get_hazard('tropical_cyclone',
                                properties={'country_name': 'Haiti',
                                            'climate_scenario': 'rcp60',
                                            'ref_year': str(future_year),
                                            'nb_synth_tracks':'10'})
```

```
2022-03-03 05:35:22,192 - climada.hazard.base - INFO - Reading /Users/chrisfairless/
↪climada/data/hazard/tropical_cyclone/tropical_cyclone_10synth_tracks_150arcsec_HTI_
↪1980_2020/v1/tropical_cyclone_10synth_tracks_150arcsec_HTI_1980_2020.hdf5
2022-03-03 05:35:28,402 - climada.hazard.base - INFO - Reading /Users/chrisfairless/
↪climada/data/hazard/tropical_cyclone/tropical_cyclone_10synth_tracks_150arcsec_
↪rcp85_HTI_2080/v1/tropical_cyclone_10synth_tracks_150arcsec_rcp85_HTI_2080.hdf5
```

We can plot the hazards and show how they are forecast to intensify. For example, showing the strength of a 50-year return period wind in present and future climates:

```
# Plot the hazards, showing 50-year return period hazard
haz_present.plot_rp_intensity(return_periods=(50,), smooth=False, vmin=32, vmax=50)
haz_future.plot_rp_intensity(return_periods=(50,), smooth=False, vmin=32, vmax=50)
```

```
2022-03-03 05:35:28,479 - climada.hazard.base - INFO - Computing exceedance intenstiy␣
↪map for return periods: [50]
2022-03-03 05:35:36,986 - climada.hazard.base - INFO - Computing exceedance intenstiy␣
↪map for return periods: [50]
```

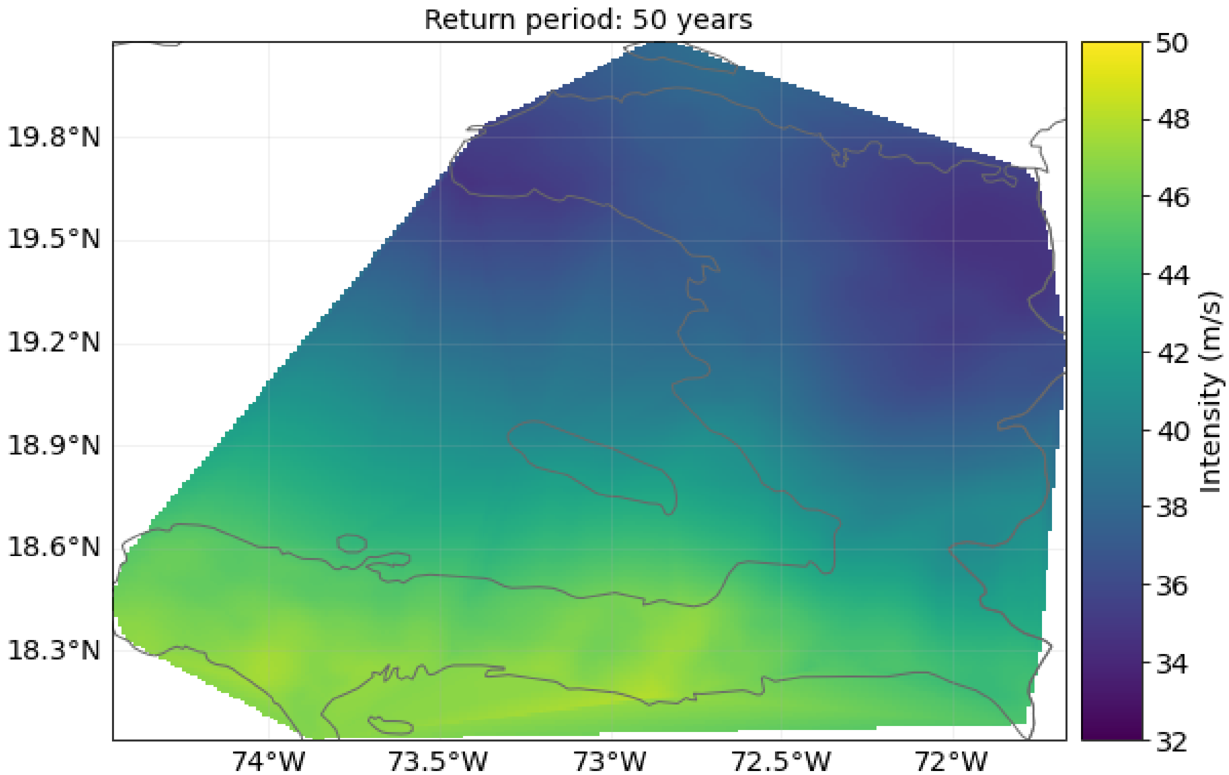
```
(<GeoAxesSubplot:title={'center':'Return period: 50 years'}>,
 array([[41.84896948, 41.98439726, 41.62016887, ..., 49.52344953,
         51.35294266, 51.51945831]]))
```

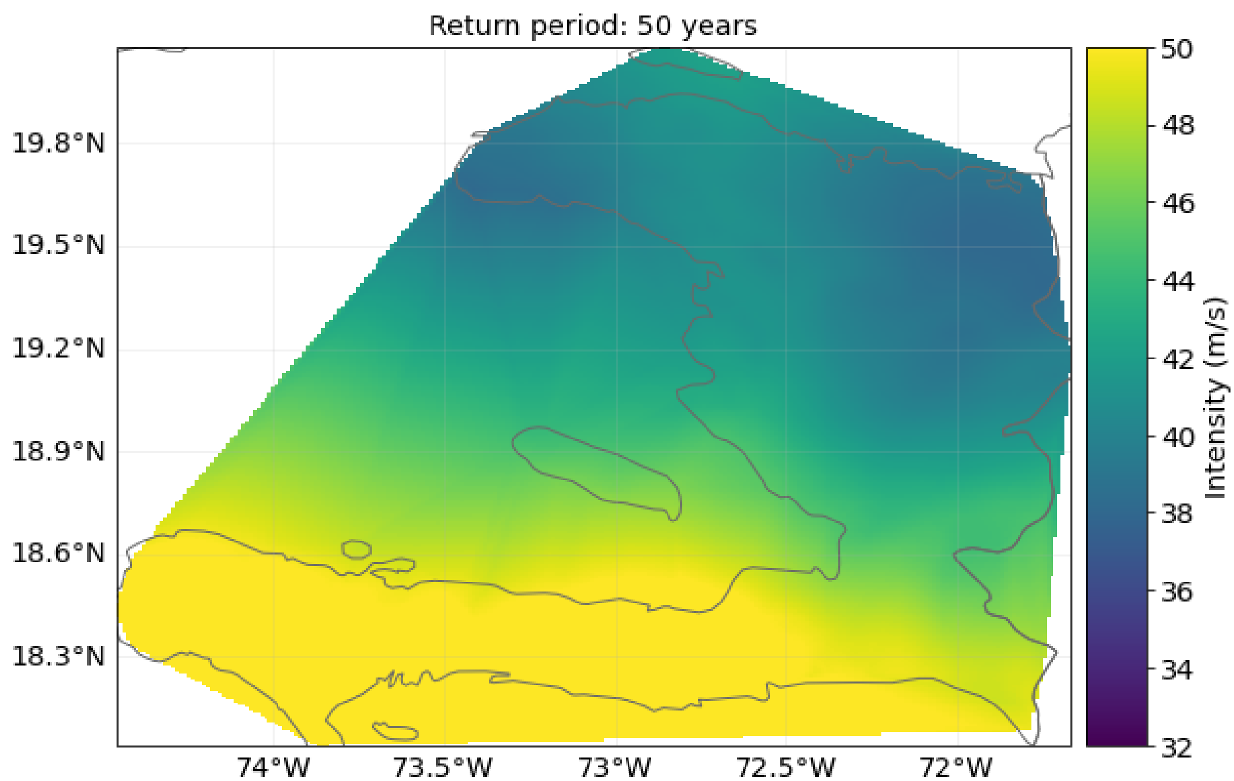
```
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:825: ShapelyDeprecationWarning: __len__ for multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Check the length of the `geoms`␣
↪property instead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:877: ShapelyDeprecationWarning: Iteration over multi-part geometries␣
↪is deprecated and will be removed in Shapely 2.0. Use the `geoms` property to␣
↪access the constituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:944: ShapelyDeprecationWarning: __len__ for multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Check the length of the `geoms`␣
```

(continues on next page)

(continued from previous page)

```
↪property instead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
```





Download LitPop economic exposure data

The Data API provides us with economic exposure data:

```
exp_present = client.get_litpop(country='Haiti')
```

```
2022-03-03 05:35:52,700 - climada.entity.exposures.base - INFO - Reading /Users/
↪chrisfairless/climada/data/exposures/litpop/LitPop_150arcsec_HTI/v1/LitPop_
↪150arcsec_HTI.hdf5
```

For 2080's economic exposure we will use a crude approximation, assuming the country will experience 2% economic growth annually:

```
import copy

exp_future = copy.deepcopy(exp_present)
exp_future.ref_year = future_year
n_years = exp_future.ref_year - exp_present.ref_year + 1
growth_rate = 1.02
growth = growth_rate ** n_years
exp_future.gdf['value'] = exp_future.gdf['value'] * growth
```

We can plot the current and future exposures. The default scale is logarithmic and we see how the values of exposures grow, though not by a full order of magnitude.

```
exp_present.plot_raster(fill=False, vmin=4, vmax=11)
exp_future.plot_raster(fill=False, vmin=4, vmax=11)
```

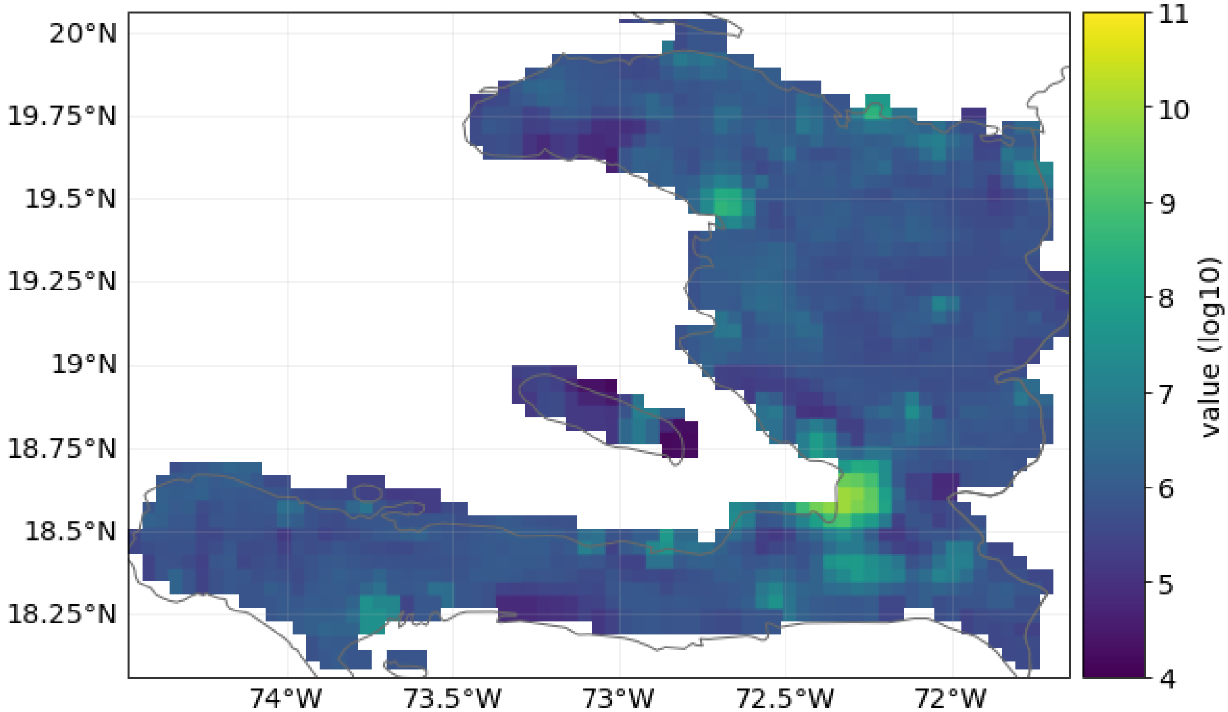
```
2022-03-03 05:35:52,895 - climada.util.coordinates - INFO - Raster from resolution 0.
↪04166666666666785 to 0.04166666666666785.
```

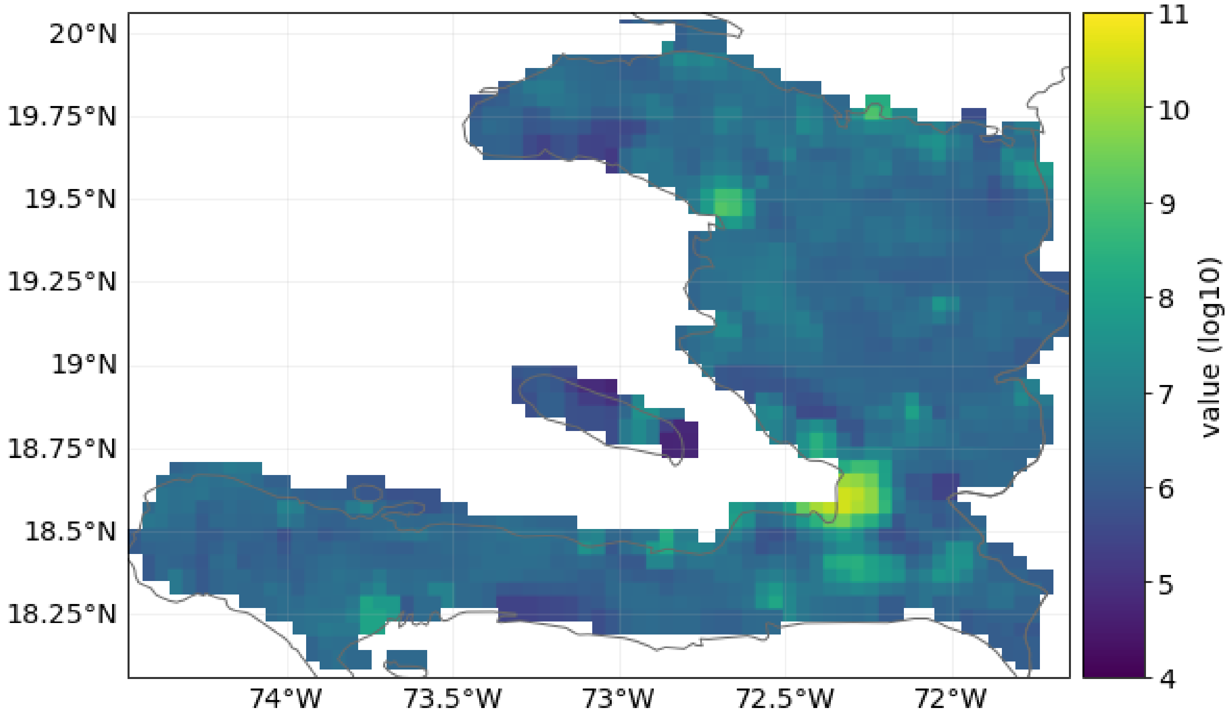
```
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪pyproj/crs/crs.py:1256: UserWarning: You will likely lose important projection␣
↪information when converting to a PROJ string from another format. See: https://proj.
↪org/faq.html#what-is-the-best-format-for-describing-coordinate-reference-systems
  return self._crs.to_proj4(version=version)
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:825: ShapelyDeprecationWarning: __len__ for multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Check the length of the `geoms`␣
↪property instead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:877: ShapelyDeprecationWarning: Iteration over multi-part geometries␣
↪is deprecated and will be removed in Shapely 2.0. Use the `geoms` property to␣
↪access the constituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:944: ShapelyDeprecationWarning: __len__ for multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Check the length of the `geoms`␣
↪property instead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
```

```
2022-03-03 05:35:59,536 - climada.util.coordinates - INFO - Raster from resolution 0.
↪04166666666666785 to 0.04166666666666785.
```

```
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪pyproj/crs/crs.py:1256: UserWarning: You will likely lose important projection␣
↪information when converting to a PROJ string from another format. See: https://proj.
↪org/faq.html#what-is-the-best-format-for-describing-coordinate-reference-systems
  return self._crs.to_proj4(version=version)
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:825: ShapelyDeprecationWarning: __len__ for multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Check the length of the `geoms`␣
↪property instead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:877: ShapelyDeprecationWarning: Iteration over multi-part geometries␣
↪is deprecated and will be removed in Shapely 2.0. Use the `geoms` property to␣
↪access the constituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:944: ShapelyDeprecationWarning: __len__ for multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Check the length of the `geoms`␣
↪property instead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
```

```
<GeoAxesSubplot:>
```





We then need to map the exposure points to the hazard centroids. (Note: we could have done this earlier before we copied the exposure, but not all analyses will have present and future exposures and hazards on the same sets of points.)

```
# This would be done automatically in Impact calculations
# but it's better to do it explicitly before the calculation
exp_present.assign_centroids(haz_present, distance='approx')
```

(continues on next page)

(continued from previous page)

```
exp_future.assign_centroids(haz_future, distance='approx')
```

```
2022-03-03 05:36:15,405 - climada.entity.exposures.base - INFO - Matching 1329␣
↪exposures with 1329 centroids.
2022-03-03 05:36:15,421 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-03-03 05:36:16,026 - climada.entity.exposures.base - INFO - Matching 1329␣
↪exposures with 1329 centroids.
2022-03-03 05:36:16,042 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
```

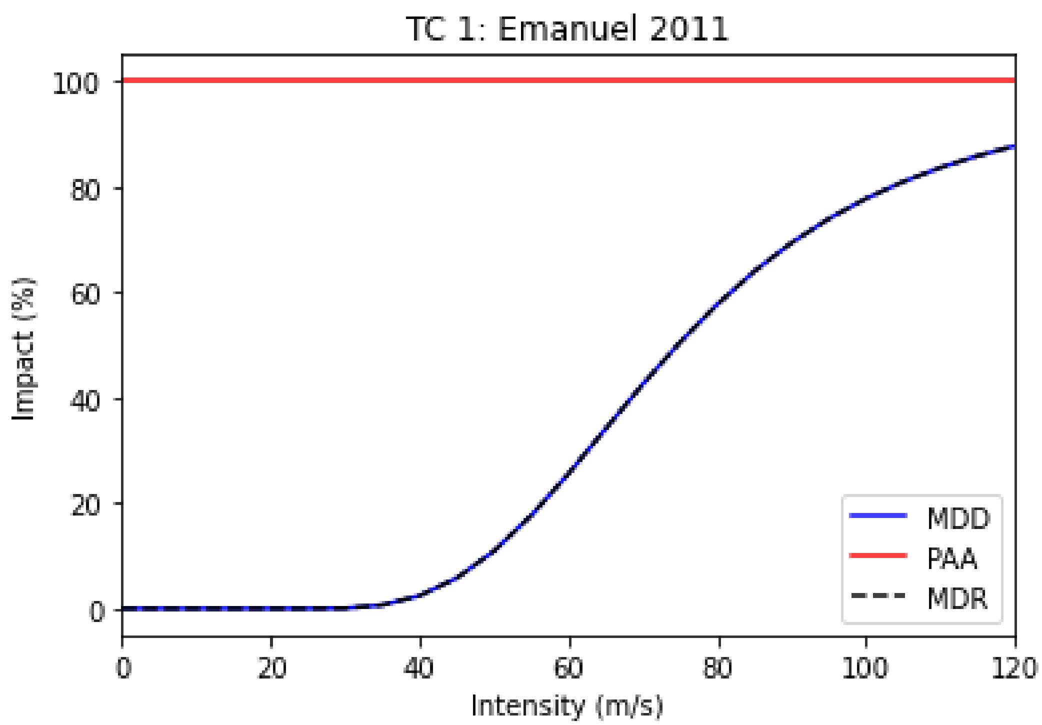
Define impact function

In this analysis we'll use the popular sigmoid curve impact function from Emanuel (2011).

```
from climada.entity import ImpactFuncSet, ImpfTropCyclone

impf_tc = ImpfTropCyclone.from_emanuel_usa()

# add the impact function to an Impact function set
impf_set = ImpactFuncSet([impf_tc])
impf_set.check()
impf_tc.plot()
```

```
2022-03-03 05:36:16,069 - climada.entity.impact_funcs.base - WARNING - For intensity␣
↪= 0, mdd != 0 or paa != 0. Consider shifting the origin of the intensity scale. In␣
↪impact.calc the impact is always null at intensity = 0.
```

```
<AxesSubplot:title={'center':'TC 1: Emanuel 2011'}, xlabel='Intensity (m/s)', ylabel=
↪'Impact (%)'>
```




```
# Rename the impact function column in the exposures and assign hazard IDs

# This is more out of politeness, since if there's only one impact function
# and one `impf_` column, CLIMADA can figure it out
exp_present.gdf.rename(columns={"impf_": "impf_TC"}, inplace=True)
exp_present.gdf['impf_TC'] = 1
exp_future.gdf.rename(columns={"impf_": "impf_TC"}, inplace=True)
exp_future.gdf['impf_TC'] = 1
```

Define adaptation measures

For adaptation measures we'll follow some of the examples from the *Adaptation MeasureSet tutorial*. See the tutorial to understand how measures work in more depth.

These numbers are completely made up. We implement one measure that reduces the (effective) wind speed by 5 m/s and one that completely protects 10% of exposed assets.

```
import numpy as np
import matplotlib.pyplot as plt
from climada.entity.measures import Measure, MeasureSet

meas_1 = Measure(
    haz_type='TC',
    name='Measure A',
    color_rgb=np.array([0.8, 0.1, 0.1]),
    cost=5000000000,
    hazard_inten_imp=(1, -5),      # Decrease wind speeds by 5 m/s
    risk_transf_cover=0,
)
```

(continues on next page)

(continued from previous page)

```
meas_2 = Measure(
    haz_type='TC',
    name='Measure B',
    color_rgb=np.array([0.1, 0.1, 0.8]),
    cost=220000000,
    paa_impact=(1, -0.10),   # 10% fewer assets affected
)

# gather all measures
meas_set = MeasureSet(measure_list=[meas_1, meas_2])
meas_set.check()
```

Define discount rates

We'll define two discount rate objects so that we can compare their effect on a cost-benefit. First, a zero discount rate, where preventing loss in 2080 is valued the same a preventing it this year. Second, the often-used 1.4% per year.

```
from climada.entity import DiscRates

year_range = np.arange(exp_present.ref_year, exp_future.ref_year + 1)
annual_discount_zero = np.zeros(n_years)
annual_discount_stern = np.ones(n_years) * 0.014

discount_zero = DiscRates(year_range, annual_discount_zero)
discount_stern = DiscRates(year_range, annual_discount_stern)
```

Create Entity objects

Now we have everything we need to create Entities. Remember, Entity is a container class for grouping ***Exposures***, ***Impact Functions***, ***Discount Rates*** and ***Measures***.

In this first example we'll set discount rates to zero.

```
from climada.entity import Entity

entity_present = Entity(exposures=exp_present, disc_rates=discount_zero,
                        impact_func_set=impf_set, measure_set=meas_set)
entity_future = Entity(exposures=exp_future, disc_rates=discount_zero,
                       impact_func_set=impf_set, measure_set=meas_set)
```

Cost-benefit #1: adaptation measures, no climate change or economic growth

We are now ready to perform our first cost-benefit analysis. We'll start with the simplest and build up complexity.

The first analysis only looks at solely at the effects of introducing adaptation measures. It assumes no climate change and no economic growth. It evaluates the benefit over the period 2018 (present) to 2080 (future) and sets the discount rate to zero.

```
from climada.engine import CostBenefit
from climada.engine.cost_benefit import risk_aai_agg

costben_measures_only = CostBenefit()
```

(continues on next page)

(continued from previous page)

```
costben_measures_only.calc(haz_present, entity_present, haz_future=None, ent_
↪future=None,
                           future_year=future_year, risk_func=risk_aai_agg, imp_time_
↪depen=None, save_imp=True)
```

```
2022-03-03 05:36:16,236 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,238 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,258 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,259 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,295 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,296 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,332 - climada.engine.cost_benefit - INFO - Computing cost benefit␣
↪from years 2018 to 2080.

Measure      Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
---------  ---------------  ------------------  --------------
Measure A             5                4.74132        0.948265
Measure B             0.22             1.10613        5.02787

--------------------  ---------  --------
Total climate risk:   11.0613    (USD bn)
Average annual risk:   0.175576  (USD bn)
Residual risk:         5.21385   (USD bn)
--------------------  ---------  --------
Net Present Values
```

Let's take a moment to look through these results.

The first table gives us a breakdown of cost-benefits by measure. We can see that, the Benefit/Cost for measure A is very just under 1, meaning that the damage prevented is slightly less than the cost of preventing it (according to the model). In comparison, the benefit of Measure B is 5 times the cost. (Note that Benefit/Cost is the inverse of Cost/Benefit: larger numbers are better).

Let's explain the three values in the second table:

- **Total climate risk:** The impact expected over the entire study period. With no changes in future hazard or exposure and no discount rates we can check that it is 63 times the next term.
- **Average annual risk:** The average annual risk without any measures implemented in the future scenario (which here is the same as the present day scenario)
- **Residual risk:** The remaining risk that hasn't been offset by the adaptation measures. Here it is the total climate risk minus the total of the 'Benefit' column of the table above it.

Combining measures

We can also combine the measures to give the cost-benefit of implementing everything:

```
combined_costben = costben_measures_only.combine_measures(['Measure A', 'Measure B'],
                                                           'Combined measures',
                                                           new_color=np.array([0.1, 0.
↪8, 0.8]),
                                                           disc_rates=discount_zero)
```

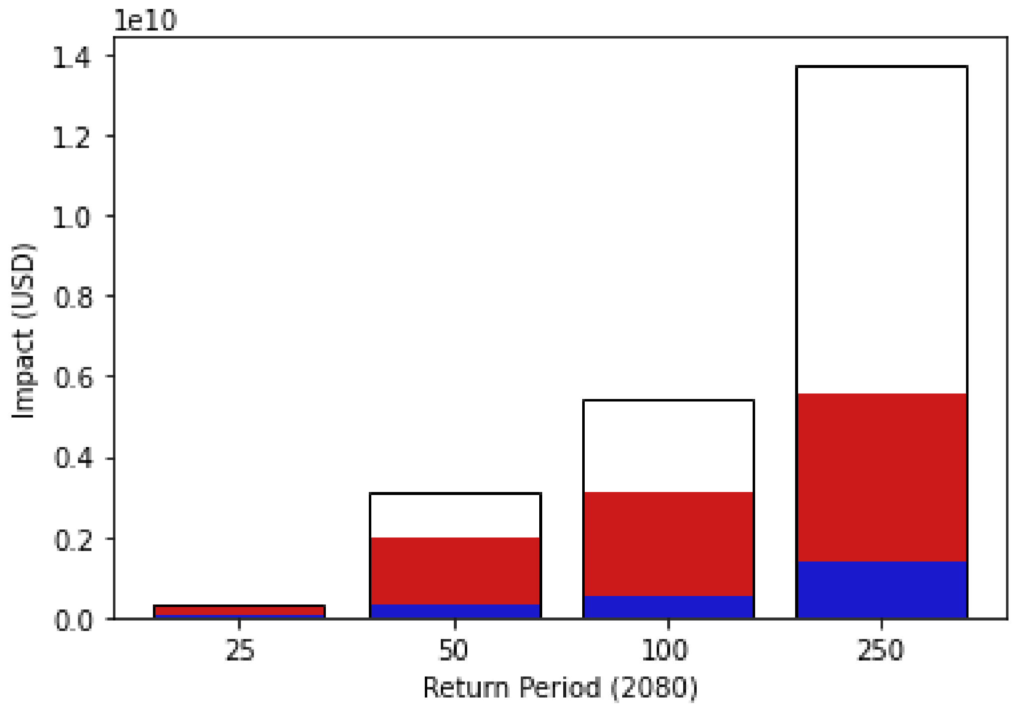
```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Combined measures             5.22             5.84344         1.11943

--------------------  ---------  --------
Total climate risk:   11.0613    (USD bn)
Average annual risk:   0.175576  (USD bn)
Residual risk:         5.21787   (USD bn)
--------------------  ---------  --------
Net Present Values
```

Note: the method of combining measures is naive. The offset impacts are summed over the event set while not letting the impact of any single event drop below zero (it therefore doesn't work in analyses where impacts can go below zero).

Plotting benefits by return period

Finally, we can see how effective the adaptation measures are at different return periods. The `plot_event_view` plot shows the difference in losses at different return periods in the future scenario (here the same as the present scenario) with the losses offset by the adaptation measures shaded.

```
ax = costben_measures_only.plot_event_view((25, 50, 100, 250))
```



We see that the Measure A, which reduces wind speeds by 5 m/s, is able to completely stop impacts at the 25 year return period, and that at 250 years – the strongest events – the measures have greatly reduced effectiveness.

Cost-benefit #2: adaptation measures with climate change and economic growth

Our next analysis will introduce a change in future scenarios. We'll add `hazard_future` and `entity_future` into the mixture. We'll set `imp_time_depen` set to 1, meaning we interpolate linearly between the present and future hazard and exposures in our summation over years. We'll still keep the discount rate at zero.

```
costben = CostBenefit()
costben.calc(haz_present, entity_present, haz_future=haz_future, ent_future=entity_
↪future,
             future_year=future_year, risk_func=risk_aai_agg, imp_time_depen=1, save_
↪imp=True)
```

```
2022-03-03 05:36:16,478 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,480 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,498 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,500 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,534 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,535 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,572 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,574 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
```

(continues on next page)

(continued from previous page)

```
2022-03-03 05:36:16,592 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,593 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:16,615 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,616 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:16,637 - climada.engine.cost_benefit - INFO - Computing cost benefit␣
↪from years 2018 to 2080.

Measure      Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
---------  ----------------  ------------------  --------------
Measure A              5               13.9728         2.79457
Measure B              0.22             3.65387       16.6085

--------------------  ---------  --------
Total climate risk:   36.5387    (USD bn)
Average annual risk:   0.984382  (USD bn)
Residual risk:        18.912     (USD bn)
--------------------  ---------  --------
Net Present Values
```

What has changed by adding climate change and population growth?

- With growing exposure and more extreme events we see about a 3-fold increase in total climate risk. Remember this is the average annual impacts summed over every year between the present and the future. The average annual risk has grown even more, by over a factor of 5. This represents the unadapted annual impacts in the future scenario.
- Greater impacts means that our adaptation measures offset more in absolute terms. The same adaptation measures create larger benefits (and therefore cost-benefits), which have increased by about a factor of three. Measure A is now clearly worth implementing in this cost-benefit analysis, whereas it wasn't before.
- We also see that the residual risk, i.e. the impacts over the analysis period that are not offset by the adaptation measures, is much larger.

Exercise: try changing the value of the `imp_time_depen` parameter in the calculation above. Values < 1 front-load the changes over time, and values > 1 back-load the changes. How does it affect the values in the printout above? What changes? What doesn't?

Waterfall plots

Now that there are more additional components in the analysis, we can use more of the CostBenefit class's visualisation methods. The waterfall plot is the clearest way to break down the components of risk:

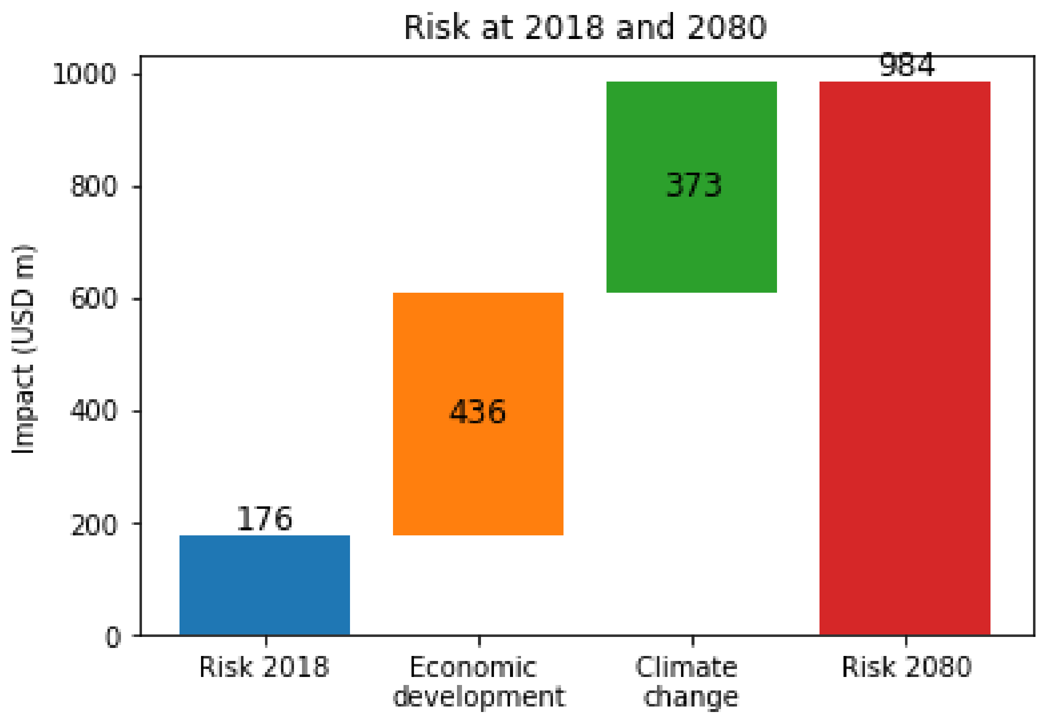
```
# define this as a function because we'll use it again later
def waterfall():
  return costben.plot_waterfall(haz_present, entity_present, haz_future, entity_
↪future,
                                risk_func=risk_aai_agg)

ax = waterfall()
```

```
2022-03-03 05:36:16,647 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
```

(continues on next page)

(continued from previous page)

```
2022-03-03 05:36:16,649 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,665 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,667 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:16,695 - climada.engine.cost_benefit - INFO - Risk at 2018: 1.756e+08
2022-03-03 05:36:16,696 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,699 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,714 - climada.engine.cost_benefit - INFO - Risk with development␣
↪at 2080: 6.113e+08
2022-03-03 05:36:16,715 - climada.engine.cost_benefit - INFO - Risk with development␣
↪and climate change at 2080: 9.844e+08
```



The waterfall plot breaks down the average annual risk faced in 2080 (this is \$0.984 bn, as printed out during the cost-benefit calculation).

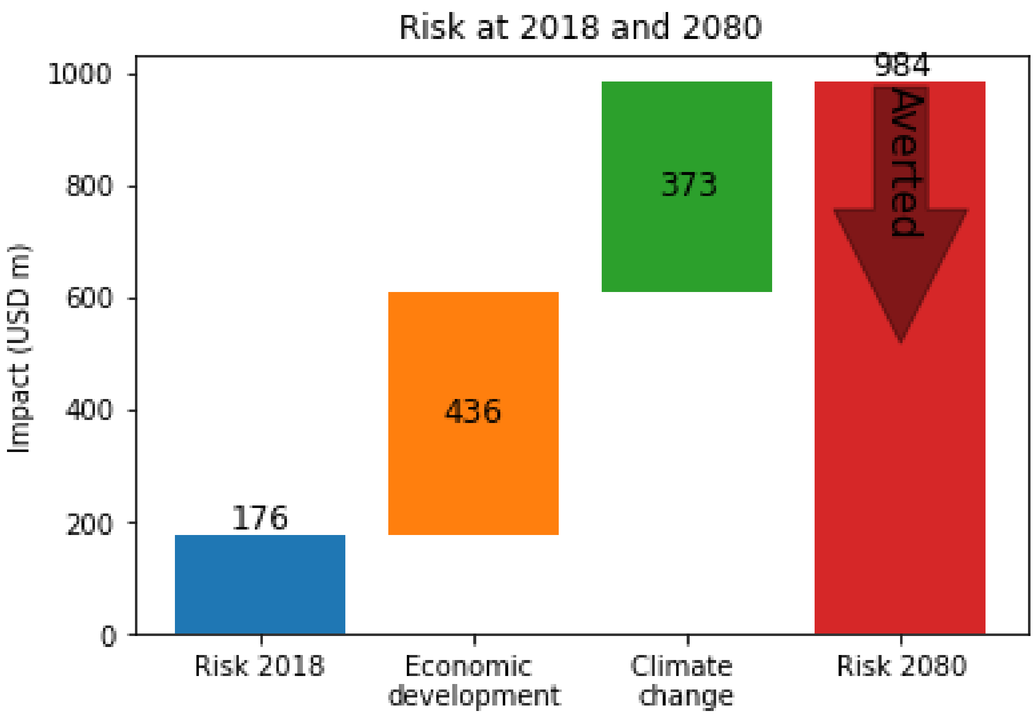
We see that the baseline 2018 risk in blue. The ‘Economic development’ bar in orange shows the change in annual impacts resulting from growth in exposure, and the ‘Climate change’ bar in green shows the additional change from changes in the hazard.

In this analysis, then, we see that changes in annual losses are likely to be driven by both economic development and climate change, in roughly equal amounts.

The `plot_arrow_averted` graph builds on this, adding an indication of the risk averted to the waterfall plot. It’s slightly awkward to use, which is why we wrote a function to create the waterfall plot earlier:

```
costben.plot_arrow_averted(axis = waterfall(), in_meas_names=['Measure A', 'Measure B
↪'], accumulate=True, combine=False,
                           risk_func=risk_aai_agg, disc_rates=None, imp_time_depen=1)
```

```
2022-03-03 05:36:16,803 - climada.engine.impact - INFO - Exposures matching centroids
↪found in centr_TC
2022-03-03 05:36:16,804 - climada.engine.impact - INFO - Calculating damage for 1329
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,820 - climada.engine.impact - INFO - Exposures matching centroids
↪found in centr_TC
2022-03-03 05:36:16,821 - climada.engine.impact - INFO - Calculating damage for 1329
↪assets (>0) and 16808 events.
2022-03-03 05:36:16,849 - climada.engine.cost_benefit - INFO - Risk at 2018: 1.756e+08
2022-03-03 05:36:16,850 - climada.engine.impact - INFO - Exposures matching centroids
↪found in centr_TC
2022-03-03 05:36:16,851 - climada.engine.impact - INFO - Calculating damage for 1329
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,866 - climada.engine.cost_benefit - INFO - Risk with development
↪at 2080: 6.113e+08
2022-03-03 05:36:16,867 - climada.engine.cost_benefit - INFO - Risk with development
↪and climate change at 2080: 9.844e+08
```



Exercise: In addition, the `plot_waterfall_accumulated` method is available to produce a waterfall plot from a different perspective. Instead of showing a breakdown of the impacts from the year of our future scenario, it accumulates the components of risk over the whole analysis period. That is, it sums the components over every year between 2018 (when the entire risk is the baseline risk) to 2080 (when the breakdown is the same as the plot above). The final plot has the same four components, but gives them different weightings. Look up the function in the `climada.engine.cost_benefit` module and try it out. Then try changing the value of the `imp_time_depen` parameter, and see how front-loading or back-loading the year-on-year changes gives different totals and different breakdowns of risk.

Cost-benefit #3: Adding discount rates

Next we will introduce discount rates to the calculations. Recall that discount rates are factors used to convert future impacts into present-day impacts, based on the idea that an impact in the future is less significant than the same impact today.

We will work with the annual 1.4% discount that we defined earlier in the `discount_stern` object. Let's define two new Entity objects with these discount rates:

```
entity_present_disc = Entity(exposures=exp_present, disc_rates=discount_stern,
                             impact_func_set=impf_set, measure_set=meas_set)
entity_future_disc = Entity(exposures=exp_future, disc_rates=discount_stern,
                            impact_func_set=impf_set, measure_set=meas_set)
```

And then re-calculate the cost-benefits:

```
costben_disc = CostBenefit()
costben_disc.calc(haz_present, entity_present_disc, haz_future=haz_future, ent_
↪future=entity_future_disc,
                  future_year=future_year, risk_func=risk_aai_agg, imp_time_depen=1,␣
↪save_imp=True)
print(costben_disc.imp_meas_future['no measure']['impact'].imp_mat.shape)
```

```
2022-03-03 05:36:16,969 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,971 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,988 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,989 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:17,024 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:17,026 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:17,062 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:17,064 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:17,079 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:17,081 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:17,100 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:17,101 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:17,123 - climada.engine.cost_benefit - INFO - Computing cost benefit␣
↪from years 2018 to 2080.

Measure      Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
---------  ---------------  ------------------  --------------
Measure A             5                8.46661         1.69332
Measure B             0.22             2.20086        10.0039


--------------------  ---------  --------
Total climate risk:   22.0086    (USD bn)
```

(continues on next page)

(continued from previous page)

```
Average annual risk:   0.984382   (USD bn)
Residual risk:        11.3412     (USD bn)
--------------------  ---------   --------
Net Present Values
(0, 0)
```

How has this changed the numbers?

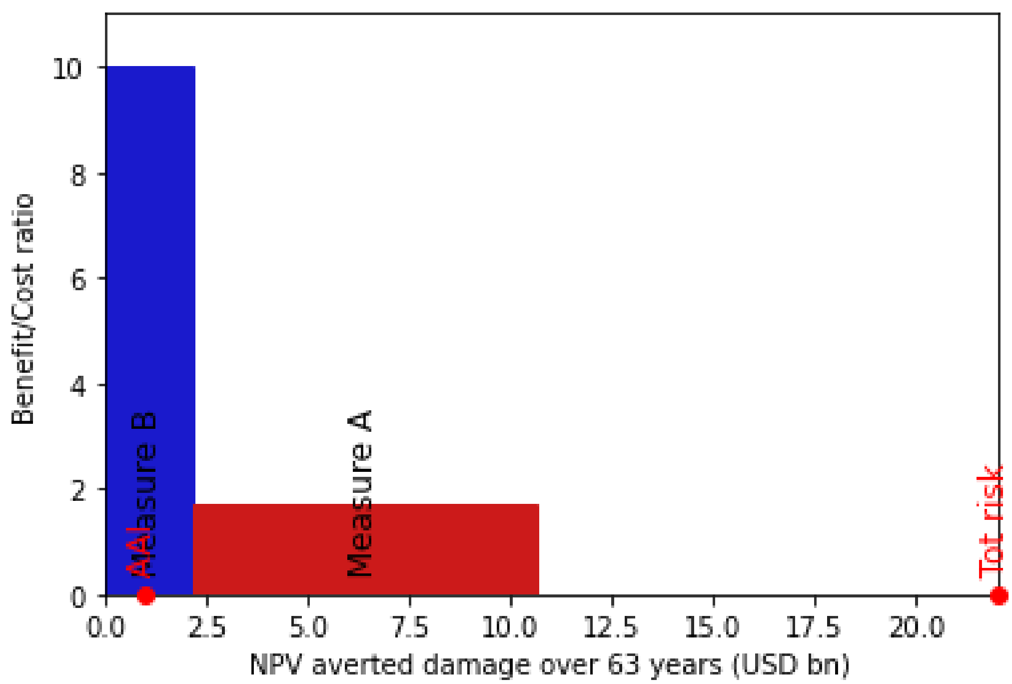
- The benefits have shrunk, since the values of the impacts prevented in the future have decreased. This means the benefit/cost ratios have decreased too. Nevertheless Measure A still has benefits that outweigh the costs.
- The total climate risk has decreased. The risk is the sum of impacts (with no adaptation measures) over the whole analysis period. Since future impacts are all smaller than they were without discount rates, the sum has decreased.
- The average annual risk, the unadapted annual risk in the future scenario, is the same (although it would be less if it was converted to a net present value).
- The residual risk has also shrunk. It has been discounted in the same way as the offset impacts.

Take together we see a slightly more optimistic outlook for climate change, as the future risks are smaller, but less attractive investments in offsetting these risks, since the benefit/cost ratio has shrunk.

Additional data exploration

With scenarios like this, the `CostBenefit.plot_cost_benefit` method shows a 2-dimensional representation of the cost-benefits.

```
ax = costben_disc.plot_cost_benefit()
```



The x-axis here is damage averted over the 2018-2080 analysis period. The y-axis is the Benefit/Cost ratio (so higher is better). This means that the area of each shape represents the total benefit of the measure. Furthermore, any measure which goes above 1 on the y-axis gives a larger benefit than the cost of its implementation.

The average annual impact and the total climate risk are marked on the x-axis. The width between the last measure bar and the the total climate risk is the residual risk.

Exercise: How sensitive are cost benefit analyses to different parameters? Let's say an adaptation measure is a 'good investment' if the benefit is greater than the cost over the analysis period, and it's a 'bad investment' if the benefit is less than the cost.

- Using the hazards and exposures from this tutorial, can you design an impact measure that is a good investment when no discount rates are applied, and a bad investment when a 1.4% (or higher) discount rate is applied?
- Create hazard and exposure objects for the same growth and climate change scenarios as this tutorial, but for the year 2040. Can you design an impact measure that is a good investment when evaluated out to 2080, but a bad investment when evaluated to 2040?
- Using the hazards and exposures from this tutorial, can you design an impact measure that is a good investment when `imp_time_depen` = 1/4 (change happens closer to 2018) and a bad investment when `imp_time_depen` = 4 (change happens closer to 2080).

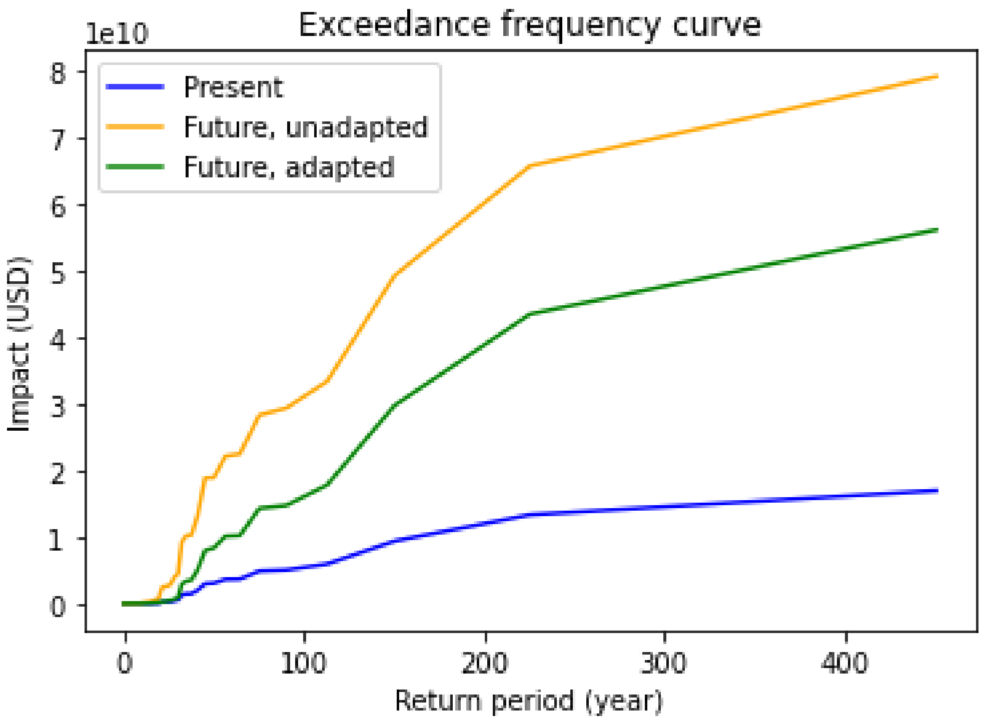
Finally we can use some of the functionality of the objects stored within the CostBenefit object. Remember that many impact calculations have been performed to get here, and if `imp_mat` was set to True, the data has been stored (or … it will be. I found a bug that stops it being saved while writing the tutorial.)

So this means that you can, for example, plot maps of return period hazard with different adaptation measures applied (or with all applied, using `combine_measures`).

Another thing to explore is exceedance curves, which are stored. Here are the curves for the present, future unadapted and future adapted scenarios:

```
combined_costben_disc = costben_disc.combine_measures(['Measure A', 'Measure B'],
                                                      'Combined measures',
                                                      new_color=np.array([0.1, 0.8, 0.
↪8]),
                                                      disc_rates=discount_stern)
efc_present = costben_disc.imp_meas_present['no measure']['efc']
efc_future = costben_disc.imp_meas_future['no measure']['efc']
efc_combined_measures = combined_costben_disc.imp_meas_future['Combined measures'][
↪'efc']

ax = plt.subplot(1, 1, 1)
efc_present.plot(axis=ax, color='blue', label='Present')
efc_future.plot(axis=ax, color='orange', label='Future, unadapted')
efc_combined_measures.plot(axis=ax, color='green', label='Future, adapted')
leg = ax.legend()
```

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Combined measures             5.22             10.6616         2.04245

--------------------  ---------  --------
Total climate risk:   22.0086    (USD bn)
Average annual risk:   0.984382  (USD bn)
Residual risk:        11.347     (USD bn)
--------------------  ---------  --------
Net Present Values
```



10.6.6 Conclusion

Cost-benefits calculations can be powerful policy tools, but they are as much an art as a science. Describing your adaptation measures well, choosing the period to evaluate them over, describing the changing climate and picking a discount rate will all affect your results and whether a particular measure is worth implementing.

Take the time to explain these choices to yourself and anyone else who wants to understand your calculations. It is also good practice to run sensitivity tests on your results: how much do your conclusions change when you use other plausible setups for the calculation?

10.7 Calculate probabilistic impact yearset

This module generates a yearly impact `yimp` object which contains probabilistic annual impacts for a specified amount of years (`sampled_years`). The impact values are extracted from a given impact `imp` object that contains impact values per event. The amount of `sampled_years` can be specified as an integer or as a list of years to be sampled for. The amount of events per sampled year (`events_per_year`) are determined with a Poisson distribution centered around n_events per year (`lam` = sum(event_impacts.frequency). Then, the probabilistic events occurring in each sampled year are sampled uniformly from the input `imp` object and summed up per year. Thus, the `yimp` object contains the sum of sampled (event) impacts for each sampled year. In contrast to the expected annual impact (eai), an `yimp` object contains an impact for EACH sampled year and this value differs among years. The number of events_per_year and the selected_events are saved in a sampling vector (`sampling_vect`).

The function impact_yearsets performs all these computational steps, taking an `imp` and the number of sampled_years (`sampled_years`) as input. The output of the function is the `yimp` object and the `sampling_vect`. Moreover, a `sampling_vect` (generated in a previous run) can be provided as optional input and the user can define `lam` and decide whether a correction factor shall be applied (the default is applying the correction factor). Reapplying the same sampling_vect does not only allow to reproduce the generated `yimp`, but also for a physically consistent way of sampling impacts caused by different hazards. The correction factor that is applied when the optional input `correction_fac=True` is a scaling of the computed `yimp` that assures that the eai(`yimp`) = eai(`imp`).

To make the process more transparent, this tutorial shows the single computations that are performed when generating an `yimp` object for a dummy event_impacts object.

```
import numpy as np

import climada.util.yearsets as yearsets
from climada.engine import Impact

# dummy event_impacts object containing 10 event_impacts with the values 10-110
# and the frequency 0.2 (Return period of 5 years)
imp = Impact()
imp.at_event = np.arange(10,110,10)
imp.frequency = np.array(np.ones(10)*0.2)

# the number of years to sample impacts for (length(yimp.at_event) = sampled_years)
sampled_years = 10

# sample number of events per sampled year
lam = np.sum(imp.frequency)
events_per_year = yearsets.sample_from_poisson(sampled_years, lam)
events_per_year
```

```
array([2, 2, 2, 0, 4, 5, 4, 2, 3, 1])
```

```
# generate the sampling vector
sampling_vect = yearsets.sample_events(events_per_year, imp.frequency)
sampling_vect
```

```
[array([8, 3]),
 array([7, 0]),
 array([4, 6]),
 array([], dtype=int32),
 array([5, 9, 1, 2]),
 array([1, 6, 0, 7, 2]),
 array([4, 9, 5, 8]),
 array([9, 8]),
 array([5, 3, 4]),
 array([1])]
```

```
# calculate the impact per year
imp_per_year = yearsets.compute_imp_per_year(imp, sampling_vect)
imp_per_year
```

```
[130, 90, 120, 0, 210, 210, 300, 190, 150, 20]
```

```
# calculate the correction factor
correction_factor = yearsets.calculate_correction_fac(imp_per_year, imp)
correction_factor
```

```
0.7746478873239436
```

```
# compare the resulting yimp with our step-by-step computation without applying the␣
↪correction factor:
yimp, sampling_vect = yearsets.impact_yearset(imp, sampled_years=list(range(1,11)),␣
↪correction_fac=False)

print('The yimp.at_event values equal our step-by-step computed imp_per_year:')
print('yimp.at_event = ', yimp.at_event)
print('imp_per_year = ', imp_per_year)
```

```
The yimp.at_event values equal our step-by-step computed imp_per_year:
yimp.at_event =  [90, 240, 150, 70, 40, 90, 60, 90, 170, 110]
imp_per_year =  [130, 90, 120, 0, 210, 210, 300, 190, 150, 20]
```

```
# and here the same comparison with applying the correction factor (default settings):
yimp, sampling_vect = yearsets.impact_yearset(imp, sampled_years=list(range(1,11)))

print('The same can be shown for the case of applying the correction factor.'
      'The yimp.at_event values equal our step-by-step computed imp_per year:')
print('yimp.at_event = ', yimp.at_event)
print('imp_per_year = ', imp_per_year/correction_factor)
```

```
The same can be shown for the case of applying the correction factor.The yimp.at_
↪event values equal our step-by-step computed imp_per year:
yimp.at_event =  [ 54.54545455  47.72727273  95.45454545 109.09090909   0.
  40.90909091  27.27272727   0.         109.09090909  27.27272727]
imp_per_year =  [167.81818182 116.18181818 154.90909091   0.         271.09090909
 271.09090909 387.27272727 245.27272727 193.63636364  25.81818182]
```

CHAPTER

ELEVEN

UNCERTAINTY QUANTIFICATION TUTORIALS

11.1 Unsequa - a module for uncertainty and sensitivity analysis

This is a tutorial for the unsequa module in CLIMADA. A detailled description can be found in Kropf (2021).

11.2 Table of Contents

· *Many scenarios of hazards and exposures*

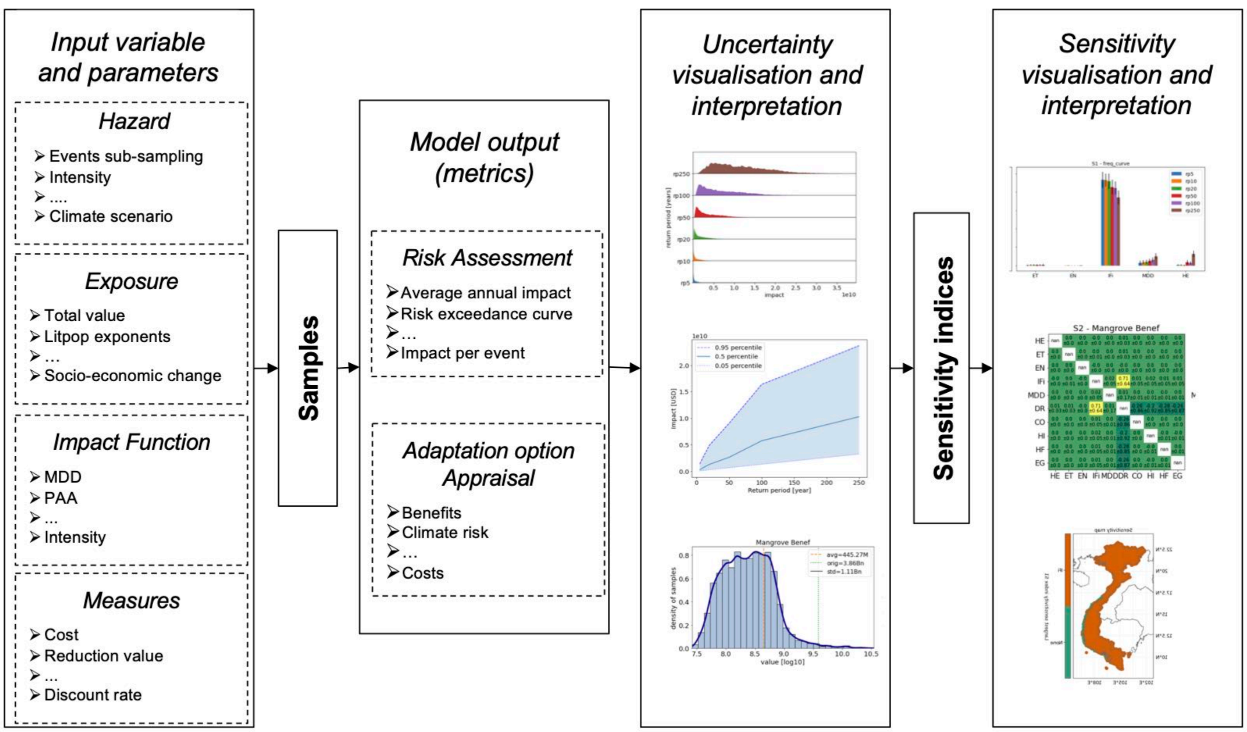
* *Input variable: Repeated loading of files made efficient*

11.2.1 Uncertainty and sensitivity analysis

Before doing an uncertainty quantification in CLIMADA, it is imperative that you get first comfortable with the different notions of uncertainty in the modelling world (see e.g.Pianosi (2016) or Douglas-Smith(2020) for a review). In particular, note that the uncertaintity values will only be as good as the input from the user. In addition, not all uncertainties can be numerically quantified, and even worse, some unkonwns are unknown. This means that sometimes, quantifying uncertainty can lead to false confidence in the output!. For a more philosophical discussion about the types of uncertainties in climate research see Knüsel (2020) and Otth (2022).

In this module, it is possible to perform global uncertainty analysis, as well as a sensitivity analysis. The word global is meant as opposition to the 'one-factor-at-a-time' (OAT) strategy. The OAT strategy, which consists in analyzing the effect of varying one model input factor at a time while keeping all other fixed, is popular among modellers, but has major shortcomings Saltelli (2010), Saltelli(2019) and should not be used.

A rough schemata of how to perform uncertainty and sensitivity analysis (taken from Kropf(2021))



1. Kropf, C.M. et al. Uncertainty and sensitivity analysis for global probabilistic weather and climate risk modelling: an implementation in the CLIMADA platform (2021)

2. Pianosi, F. et al. Sensitivity analysis of environmental models: A systematic review with practical workflow. Environmental Modelling & Software 79, 214–232 (2016). 3.Douglas-Smith, D., Iwanaga, T., Croke, B. F. W. & Jakeman, A. J. Certain trends in uncertainty and sensitivity analysis: An overview of software tools and techniques. Environmental Modelling & Software 124, 104588 (2020)

3. Knüsel, B. Epistemological Issues in Data-Driven Modeling in Climate Research. (ETH Zurich, 2020)

4. Saltelli, A. et al. Why so many published sensitivity analyses are false: A systematic review of sensitivity analysis practices. Environmental Modelling & Software 114, 29–39 (2019)
5. Saltelli, A. & Annoni, P. How to avoid a perfunctory sensitivity analysis. Environmental Modelling & Software 25, 1508–1517 (2010)

11.2.2 Unsequa Module Structure

The unsequa module contains several key classes.

The model input parameters and their distribution are specified as

- `InputVar`: defines input uncertainty variables

The input parameter sampling, Monte-Carlo uncertainty distribution calculation and the sensitivity index computation are done in

- `CalcImpact`: compute uncertainties for outputs of `climada.engine.impact.calc` (child class of `Calc`)
- `CalcDeltaImpact`: compute uncertainties for outputs of `climada.engine.impact.calc` (child class of `Calc`)
- `CalcCostBenefit`: compute uncertainties for outputs of `climada.engine.cost_benefit.calc` (child class of `Calc`)

The results are stored in

- `UncOutput`: store the uncertainty and sensitivity analysis results. Contains also several plotting methods. This is a class which only stores data.
- `UncImpactOutput`: subclass with dataframes specifically for `climada.engine.impact.calc` uncertainty and sensitivity analysis results.
- `UncCostBenefitOutput`: subclass with dataframes specifically for `climada.engine.cost_benefit.calc` uncertainty and sensitivity analysis results.

11.2.3 InputVar

The InputVar class is used to define uncertainty variables.

Attribute	Type	Description
func	function	Model variable defined as a function of the uncertainty input parameters
distr_dict	dict	Dictionary of the probability density distributions of the uncertainty input parameters

An **input uncertainty parameter** is a numerical input value that has a certain probability density distribution in your model, such as the total exposure asset value, the slope of the vulnerability function, the exponents of the litpop exposure, the value of the discount rate, the cost of an adaptation measure, …

The probability density distributions (values of `distr_dict`) of the input uncertainty parameters (keyword arguments of the `func` and keys of the `distr_dict`) can be any of the ones defined in scipy.stats.

Several helper methods exist to make generic InputVar for Exposures, ImpactFuncSet, Hazard, Entity (including DiscRates and Measures). These are described in details in the tutorial *Helper methods for InputVar*. These are a good bases for your own computations.

Example - custom continuous uncertainty parameter

Suppose we assume that the GDP value used to scale the exposure has a relative error of +-10%.

```
import warnings
warnings.filterwarnings('ignore') #Ignore warnings for making the tutorial's pdf.

#Define the base exposure
from climada.util.constants import EXP_DEMO_H5
from climada.entity import Exposures
exp_base = Exposures.from_hdf5(EXP_DEMO_H5)
```

```
# Define the function that returns an exposure with scaled total assed value
# Here x_exp is the input uncertainty parameter and exp_func the inputvar.func.
def exp_func(x_exp, exp_base=exp_base):
    exp = exp_base.copy()
    exp.gdf.value *= x_exp
    return exp
```

```
# Define the Uncertainty Variable with +-10% total asset value
# The probability density distribution of the input uncertainty parameter x_exp is sp.
↪stats.uniform(0.9, 0.2)
from climada.engine.unsequa import InputVar
import scipy as sp

exp_distr = {"x_exp": sp.stats.uniform(0.9, 0.2),
              }
exp_iv = InputVar(exp_func, exp_distr)
```

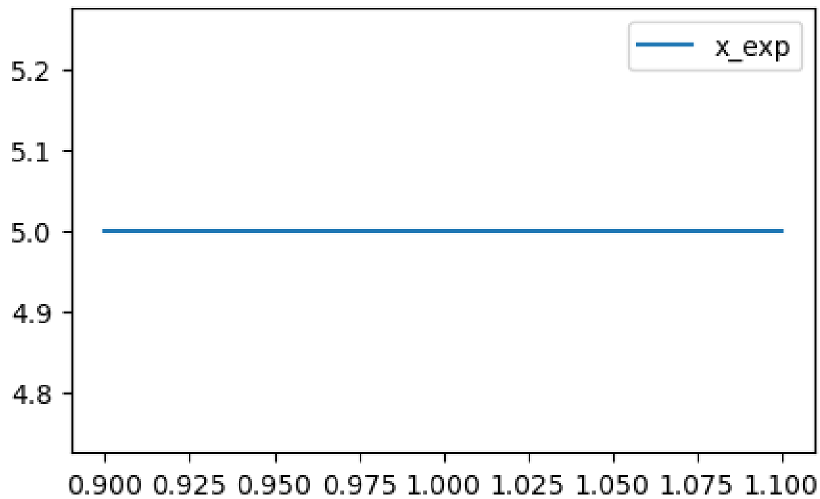
```
# Uncertainty parameters
exp_iv.labels
```

```
['x_exp']
```

```
# Evaluate for a given value of the uncertainty parameters
exp095 = exp_iv.func(x_exp = 0.95)
print(f"Base value is {exp_base.gdf['value'].sum()}, and the value for x_exp=0.95 is
↪{exp095.gdf['value'].sum()}")
```

```
Base value is 657053294559.9105, and the value for x_exp=0.95 is 624200629831.9148
```

```
# Defined distribution
exp_iv.plot(figsize=(5, 3));
```



Example - custom categorical uncertainty parameter

Suppose we want to test different exponents (m=1,2 ; n=1,2) for the LitPop exposure for the country Switzerland.

```
from climada.entity import LitPop

m_min, m_max = (1, 2)
n_min, n_max = (1, 2)

# Define the function
# Note that this here works, but might be slow because the method LitPop is called␣
↪everytime the the function
# is evaluated, and LitPop is relatively slow.
def litpop_cat(m, n):
    exp = Litpop.from_countries('CHE', res_arcsec=150, exponent=[m, n])
    return exp
```

```
# A faster method would be to first create a dictionnary with all the exposures. This␣
↪however
# requires more memory and precomputation time (here ~3-4mins)
exp = LitPop()
litpop_dict = {}
for m in range(m_min, m_max + 1):
    for n in range(n_min, n_max + 1):
        exp_mn = LitPop.from_countries('CHE', res_arcsec=150, exponents=[m, n]);
        litpop_dict[(m, n)] = exp_mn

def litpop_cat(m, n, litpop_dict=litpop_dict):
    return litpop_dict[(m, n)]
```

```
#Define the distribution dictionnary
import scipy as sp
from climada.engine.unsequa import InputVar

distr_dict = {
```

(continues on next page)

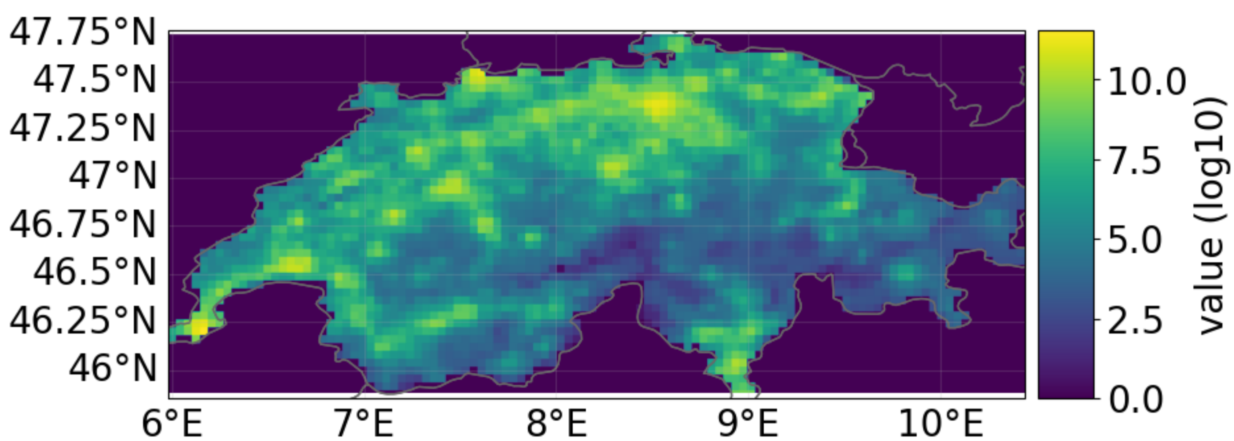
(continued from previous page)

```
    'm': sp.stats.randint(low=m_min, high=m_max+1),
    'n': sp.stats.randint(low=n_min, high=n_max+1)
    }

cat_iv = InputVar(litpop_cat, distr_dict) # One can use either of the above␣
↪definitions of litpop_cat
```

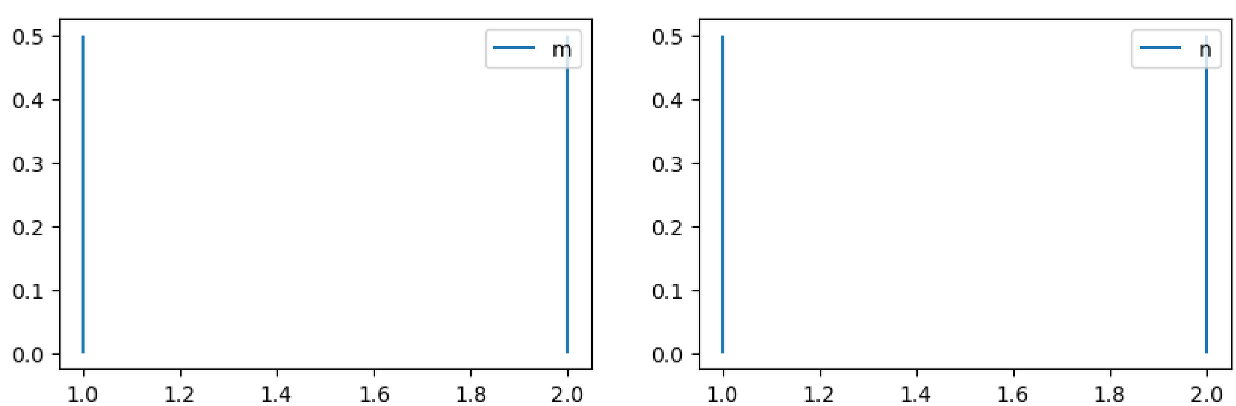
```
# Uncertainty parameters
cat_iv.labels
```

```
['m', 'n']
```

```
cat_iv.evaluate(m=1, n=2).plot_raster();
```




```
cat_iv.plot(figsize=(10, 3));
```



11.2.4 UncOutput

The `UncOutput` class is used to store data from sampling, uncertainty and sensitivity analysis. An UncOutput object can be saved and loaded from .hdf5. The classes `UncImpactOuput` and `UncCostBenefitOutput` are extensions of `UncOutput` specific for `CalcImpact` and `CalcCostBenefit`, respectively.

Data attributes

Attribute	Type	Description
samples_df	pandas.dataframe	Each row represents a sample obtained from the input parameters (one per column) distributions
UncImpactOutput		
aai_agg_unc_df	pandas.dataframe	Uncertainty data for `aai_agg`
freq_curve_unc_df	pandas.dataframe	Uncertainty data for `freq_curve`. One return period per column.
eai_exp_unc_df	pandas.dataframe	Uncertainty data for `eai_exp`. One exposure point per column.
at_event_unc_df	pandas.dataframe	Uncertainty data for `at_event`. One event per column.
UncCostBenefitOutput		
imp_meas_present_un	pandas.dataframe	Uncertainty data for `imp_meas_present`. One measure per column.
imp_meas_future_unc_	pandas.dataframe	Uncertainty data for `imp_meas_present`. One measure per column
tot_climate_risk_unc_	pandas.dataframe	Uncertainty data for `tot_climate_risk`. One measure per column.
benefit_unc_df	pandas.dataframe	Uncertainty data for `benefit`. One measure per column.
cost_ben_ratio_unc_df	pandas.dataframe	Uncertainty data for `cost_ben_ratio`. One measure per column.
cost_benefit_kwargs	dictionary	Keyword arguments for `climada.engine.cost_benefit.calc`.

Metadata and input data attributes

These attributes are used for book-keeping and characterize the sample, uncertainty and sensitivity data. These attributes are set by the methods from classes `CalcImpact` and `CalcCostBenefit` used to generate sample, uncertainty and sensitivity data.

Attribute	Type	Description
sampling_met	str	The sampling method as defined in SALib. Possible choices: 'saltelli', 'fast_sampler', 'latin', 'morris', 'dgsm', 'ff'
sampling_kwa	dict	Keyword arguments for the sampling_method.
n_samples	int	Effective number of samples (number of rows of samples_df)
param_lat	list(st	Name of all the uncertainty input parameters
problem_sa	dict	The description of the uncertainty variables and their distribution as used in SALib.
sensitivity_metho	str	Sensitivity analysis method from SALib.analyse Possible choices: 'fast', 'rbd_fact', 'morris', 'sobol', 'delta', 'ff'. Note that in Salib, sampling methods and sensitivity analysis methods should be used in specific pairs.
sensitivity_kwarg	dict	Keyword arguments for sensitivity_method.
unit	str	Unit of the exposures value

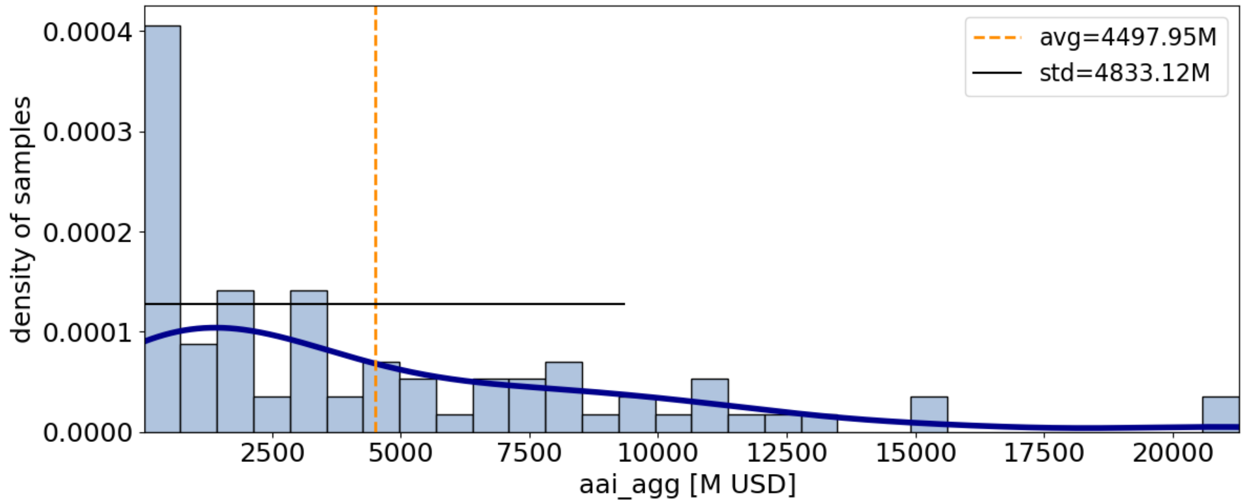
Example from file

Here we show an example loaded from file. In the sections below this class is extensively used and further examples can be found.

```
# Download the test file from the API
# Requires internet connection
from climada.util.constants import TEST_UNC_OUTPUT_IMPACT
from climada.util.api_client import Client
apiclient = Client()
ds = apiclient.get_dataset_info(name=TEST_UNC_OUTPUT_IMPACT, status='test_dataset')
_target_dir, [filename] = apiclient.download_dataset(ds)
```

```
# If you produced your own data, you do not need the API. Just replace 'filename'␣
↪with the path to your file.
from climada.engine.unsequa import UncOutput
unc_imp = UncOutput.from_hdf5(filename)
```

```
unc_imp.plot_uncertainty(metric_list=['aai_agg'], figsize=(12,5));
```




```
# Download the test file from the API
# Requires internet connection
from climada.util.constants import TEST_UNC_OUTPUT_COSTBEN
from climada.util.api_client import Client
apiclient = Client()
ds = apiclient.get_dataset_info(name=TEST_UNC_OUTPUT_COSTBEN, status='test_dataset')
_target_dir, [filename] = apiclient.download_dataset(ds)
```

```
# If you produced your own data, you do not need the API. Just replace 'filename'␣
↪with the path to your file.
from climada.engine.unsequa import UncOutput
unc_cb = UncOutput.from_hdf5(filename)
```

```
unc_cb.get_uncertainty().tail()
```

```
    Mangroves Benef  Beach nourishment Benef  Seawall Benef  \
35     2.375510e+08             1.932608e+08  234557.682554
36     9.272772e+07             7.643803e+07    9554.257314
37     1.464219e+08             1.179927e+08  192531.748810
38     9.376369e+07             7.722882e+07   10681.112247
39     9.376369e+07             7.722882e+07   10681.112247

    Building code Benef  Mangroves CostBen  Beach nourishment CostBen  \
35         1.584398e+08           6.347120                  10.277239
36         5.501366e+07          16.260133                  25.984286
37         8.979471e+07          10.297402                  16.833137
38         5.555413e+07          12.965484                  20.736269
39         5.555413e+07          16.080478                  25.718218

    Seawall CostBen  Building code CostBen  no measure - risk - future  \
35     4.350910e+04              66.742129                6.337592e+08
36     1.068151e+06             192.217876                2.200547e+08
37     5.300629e+04             117.764285                3.591788e+08
38     7.703765e+05             153.475031                2.222165e+08
39     9.554617e+05             190.347852                2.222165e+08

    no measure - risk_transf - future  ...  \
35                                0.0  ...
```

(continues on next page)

(continued from previous page)

```
36                              0.0  ...
37                              0.0  ...
38                              0.0  ...
39                              0.0  ...

    Beach nourishment - cost_ins - future  Seawall - risk - future  \
35                                     1             6.335246e+08
36                                     1             2.200451e+08
37                                     1             3.589863e+08
38                                     1             2.222058e+08
39                                     1             2.222058e+08

    Seawall - risk_transf - future  Seawall - cost_meas - future  \
35                               0                 1.020539e+10
36                               0                 1.020539e+10
37                               0                 1.020539e+10
38                               0                 8.228478e+09
39                               0                 1.020539e+10

    Seawall - cost_ins - future  Building code - risk - future  \
35                            1                   4.753194e+08
36                            1                   1.650410e+08
37                            1                   2.693841e+08
38                            1                   1.666624e+08
39                            1                   1.666624e+08

    Building code - risk_transf - future  Building code - cost_meas - future  \
35                                     0                        1.057461e+10
36                                     0                        1.057461e+10
37                                     0                        1.057461e+10
38                                     0                        8.526172e+09
39                                     0                        1.057461e+10

    Building code - cost_ins - future  tot_climate_risk
35                                  1      6.337592e+08
36                                  1      2.200547e+08
37                                  1      3.591788e+08
38                                  1      2.222165e+08
39                                  1      2.222165e+08

[5 rows x 29 columns]
```

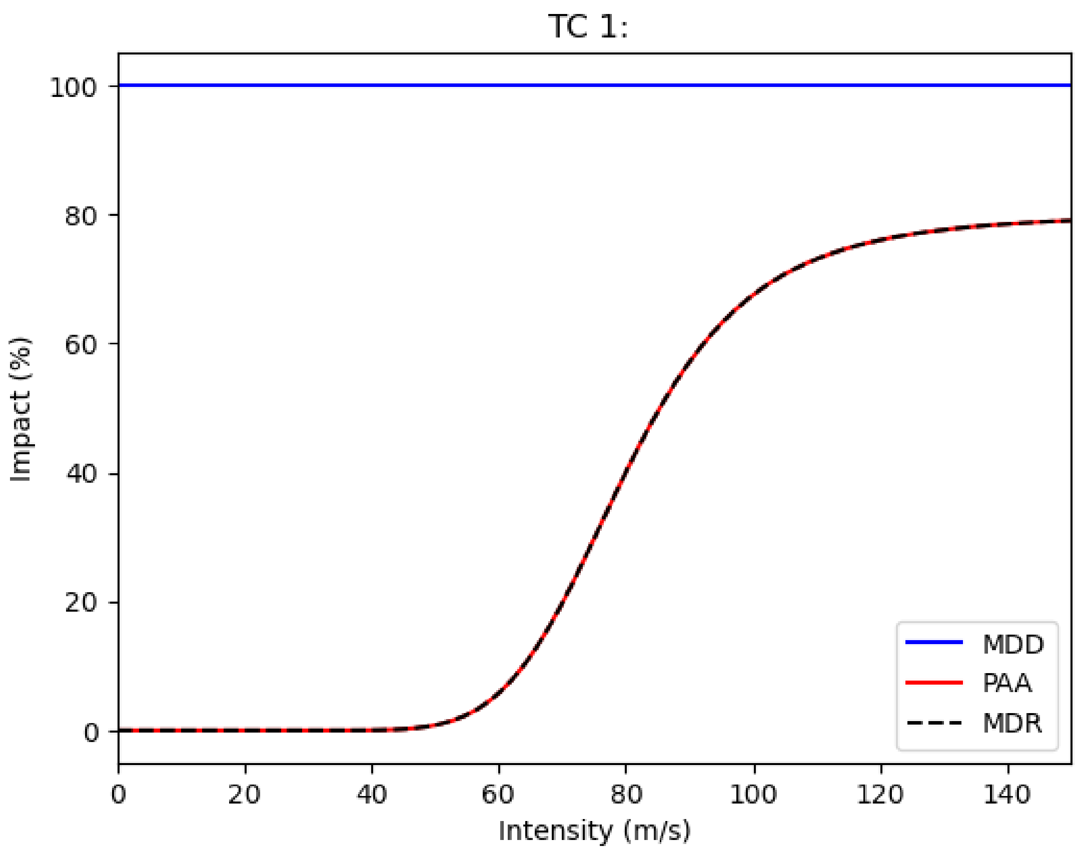
11.2.5 CalcImpact

Set the InputVars

In this example, we model the impact function for tropical cyclones on the parametric function suggested in Emanuel (2015) with 4 parameters. The exposures total value varies between 80% and 120%. For that hazard, we assume to have no good error estimate and thus do not define an InputVar for the hazard.

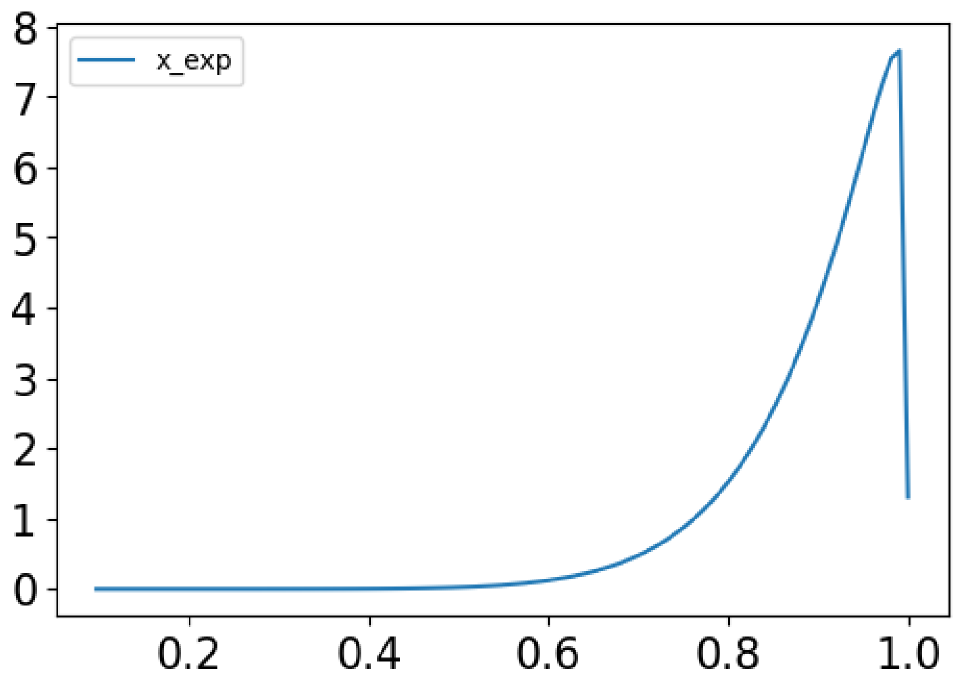
```
#Define the input variable functions
import numpy as np

from climada.entity import ImpactFunc, ImpactFuncSet, Exposures
from climada.util.constants import EXP_DEMO_H5, HAZ_DEMO_H5
from climada.hazard import Hazard

def impf_func(G=1, v_half=84.7, vmin=25.7, k=3, _id=1):

    def xhi(v, v_half, vmin):
        return max([(v - vmin), 0]) / (v_half - vmin)

    def sigmoid_func(v, G, v_half, vmin, k):
        return G * xhi(v, v_half, vmin)**k / (1 + xhi(v, v_half, vmin)**k)

    #In-function imports needed only for parallel computing on Windows
    import numpy as np
    from climada.entity import ImpactFunc, ImpactFuncSet
    intensity_unit = 'm/s'
    intensity = np.linspace(0, 150, num=100)
    mdd = np.repeat(1, len(intensity))
    paa = np.array([sigmoid_func(v, G, v_half, vmin, k) for v in intensity])
    imp_fun = ImpactFunc("TC", _id, intensity, mdd, paa, intensity_unit)
    imp_fun.check()
    impf_set = ImpactFuncSet([imp_fun])
    return impf_set

haz = Hazard.from_hdf5(HAZ_DEMO_H5)
exp_base = Exposures.from_hdf5(EXP_DEMO_H5)
#It is a good idea to assign the centroids to the base exposures in order to avoid
↪repeating this
# potentially costly operation for each sample.
exp_base.assign_centroids(haz)
def exp_base_func(x_exp, exp_base):
    exp = exp_base.copy()
    exp.gdf.value *= x_exp
    return exp
from functools import partial
exp_func = partial(exp_base_func, exp_base=exp_base)
```

```
# Visualization of the parametrized impact function
impf_func(G=0.8, v_half=80, vmin=30,k=5).plot();
```




```
#Define the InputVars

import scipy as sp
from climada.engine.unsequa import InputVar

exp_distr = {"x_exp": sp.stats.beta(10, 1.1)} #This is not really a reasonable␣
↪distribution but is used
                                              #here to show that you can use any␣
↪scipy distribution.

exp_iv = InputVar(exp_func, exp_distr)

impf_distr = {
    "G": sp.stats.truncnorm(0.5, 1.5),
    "v_half": sp.stats.uniform(35, 65),
    "vmin": sp.stats.uniform(0, 15),
    "k": sp.stats.uniform(1, 4)
    }
impf_iv = InputVar(impf_func, impf_distr)
```

```
import matplotlib.pyplot as plt
ax = exp_iv.plot(figsize=(6,4));
plt.yticks(fontsize=16);
plt.xticks(fontsize=16);
```



Compute uncertainty and sensitivity using default methods

First, we define the UncImpact object with our uncertainty variables.

```
from climada.engine.unsequa import CalcImpact

calc_imp = CalcImpact(exp_iv, impf_iv, haz)
```

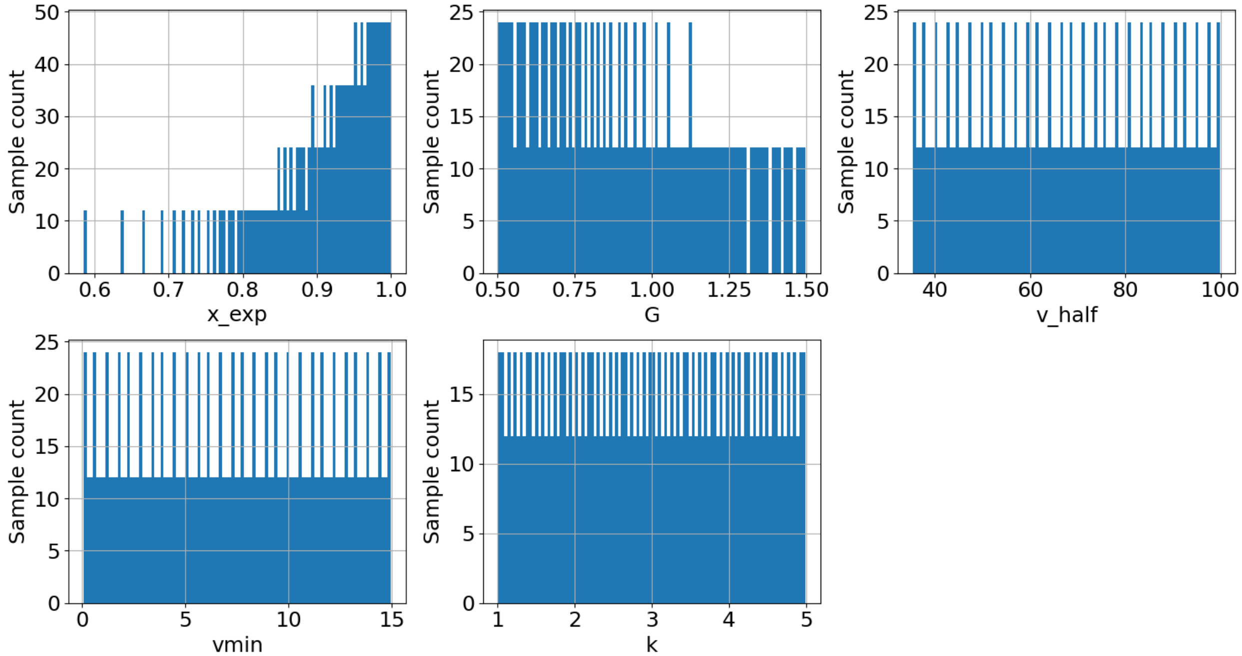
Next, we generate samples for the uncertainty parameters using the default methods. Note that depending on the chosen Salib method, the effective number of samples differs from the input variable N. For the default 'saltelli', with `calc_second_order=True`, the effective number is N(2D+2), with D the number of uncertainty parameters. See SAlib for more information.

```
output_imp = calc_imp.make_sample(N=2**7, sampling_kwargs={'skip_values': 2**8})
output_imp.get_samples_df().tail()
```

```
         x_exp         G     v_half       vmin         k
1531  0.876684  1.242977  53.662109   2.080078  4.539062
1532  0.876684  0.790617  44.013672   2.080078  4.539062
1533  0.876684  0.790617  53.662109  13.681641  4.539062
1534  0.876684  0.790617  53.662109   2.080078  3.960938
1535  0.876684  0.790617  53.662109   2.080078  4.539062
```

The resulting samples can be visualized in plots.

```
output_imp.plot_sample(figsize=(15,8));
```



Now we can compute the value of the impact metrics for all the samples. In this example, we additionaly chose to restrict the return periods 50, 100, and 250 years. By default, `eai_exp` and `at_event` are not stored.

```
output_imp = calc_imp.uncertainty(output_imp, rp = [50, 100, 250])
```

The distributions of metrics ouputs are stored as dictionaries of pandas dataframe. The metrics are directly taken from the output of `climada.impact.calc`. For each metric, on dataframe is made.

```
#All the computed uncertainty metrics attribute
output_imp.uncertainty_metrics
```

```
['aai_agg', 'freq_curve']
```

```
#One uncertainty dataframe
output_imp.get_unc_df('aai_agg').tail()
```

```
           aai_agg
1531  2.905571e+09
1532  3.755172e+09
1533  1.063119e+09
1534  2.248718e+09
1535  1.848139e+09
```

Accessing the uncertainty is in general done via the method `get_uncertainty()`. If none are specified, all metrics are returned.

```
output_imp.get_uncertainty().tail()
```

```
           aai_agg          rp50         rp100         rp250
1531  2.905571e+09  8.324391e+10  1.162643e+11  1.510689e+11
1532  3.755172e+09  1.096005e+11  1.460838e+11  1.809413e+11
1533  1.063119e+09  2.892734e+10  4.720869e+10  6.807561e+10
```

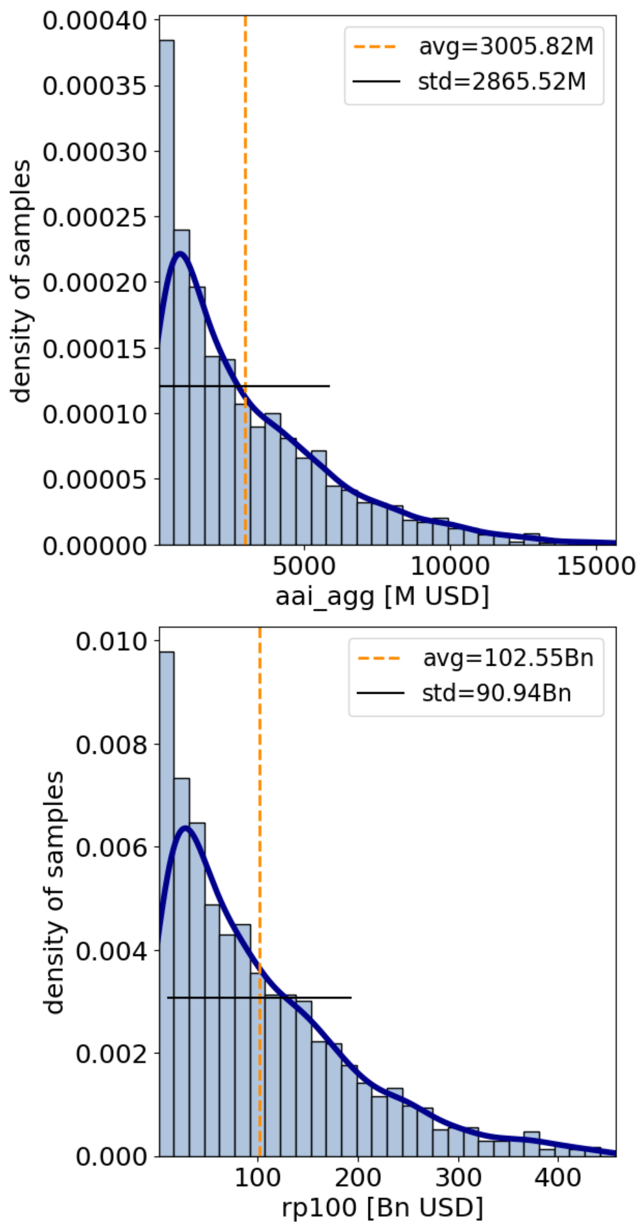
(continues on next page)

(continued from previous page)

```
1534  2.248718e+09  6.468855e+10  8.653474e+10  1.085266e+11
1535  1.848139e+09  5.294874e+10  7.395191e+10  9.609003e+10
```

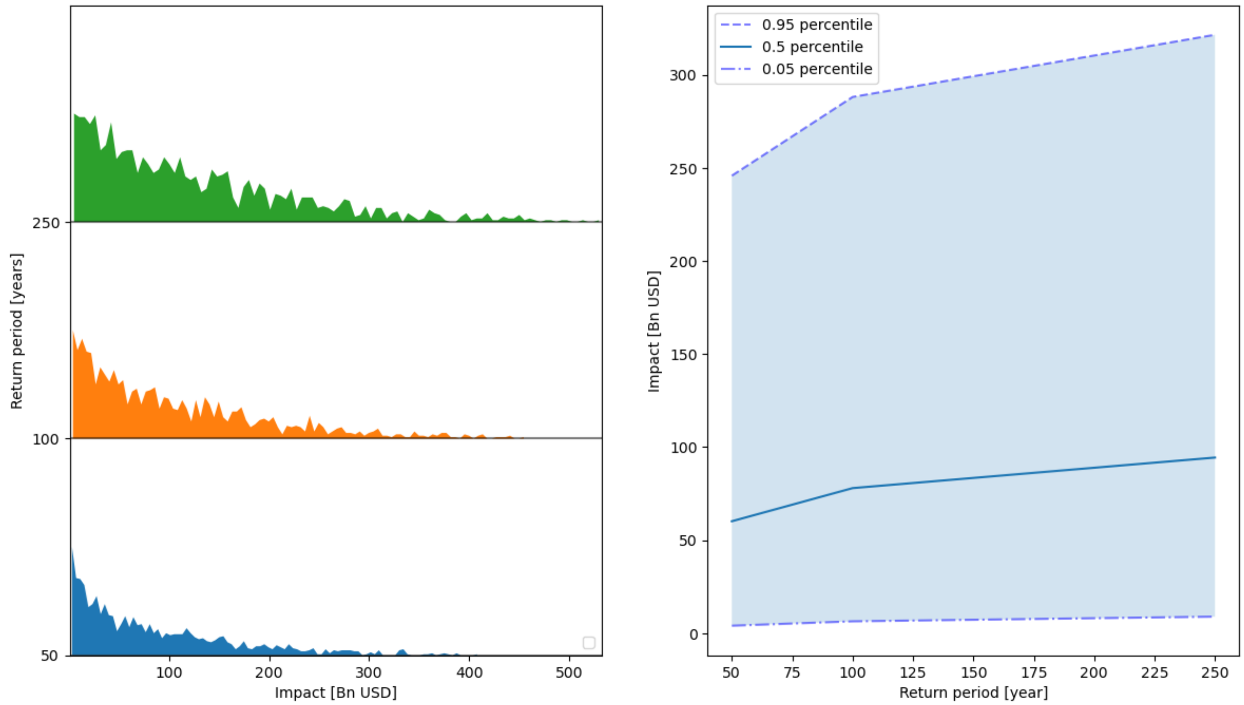
The distributions of the one-dimensioanl metrics (`eai_exp` and `at_event` are never shown with this method) can be vizualised with plots.

```
output_imp.plot_uncertainty(figsize=(12,12));
```




```
# Specific plot for the return period distributions
output_imp.plot_rp_uncertainty(figsize=(14.3,8));
```

```
No artists with labels found to put in legend.  Note that artists whose label start
↪with an underscore are ignored when legend() is called with no argument.
```



Now that a distribution of the impact metrics has been computed for each sample, we can also compute the sensitivity indices for each metrics to each uncertainty parameter. Note that the chosen method for the sensitivity analysis should correpond to its sampling partner as defined in the SAlib package.

The sensitivity indices dictionnaries outputs from the SAlib methods are stored in the same structure of nested dictionnaries as the metrics distributions. Note that depending on the chosen sensitivity analysis method the returned indices dictionnary will return specific types of sensitivity indices with specific names. Please get familiar with SAlib for more information.

Note that in our case, several of the second order sensitivity indices are negative. For the default method `sobol`, this indicates that the algorithm has not converged and cannot give realiable values for these sensitivity indices. If this happens, please use a larger number of samples. Here we will focus on the first-order indices.

```
output_imp = calc_imp.sensitivity(output_imp)
```

Similarly to the uncertainty case, the data is stored in dataframe attributes.

```
output_imp.sensitivity_metrics
```

```
['aai_agg', 'freq_curve']
```

```
output_imp.get_sens_df('aai_agg').tail()
```

```
         si param  param2  aai_agg
65  S2_conf     k   x_exp      NaN
66  S2_conf     k       G      NaN
67  S2_conf     k  v_half      NaN
68  S2_conf     k    vmin      NaN
69  S2_conf     k       k      NaN
```

To obtain the sensitivity interms of a particular sensitivity index, use the method `get_sensisitivity()`. If none is specified, the value of the index for all metrics is returned.

```
output_imp.get_sensitivity('S1')
```

```
   si    param param2   aai_agg      rp50     rp100     rp250
0  S1    x_exp   None  0.001040  0.000993  0.000930  0.001150
1  S1        G   None  0.073408  0.075781  0.084662  0.093718
2  S1   v_half   None  0.514220  0.553640  0.596659  0.619366
3  S1     vmin   None  0.012642  0.014407  0.012068  0.010065
4  S1        k   None  0.213491  0.189862  0.134867  0.095861
```

Sometimes, it is useful to simply know what is the largest sensitivity index for each metric.

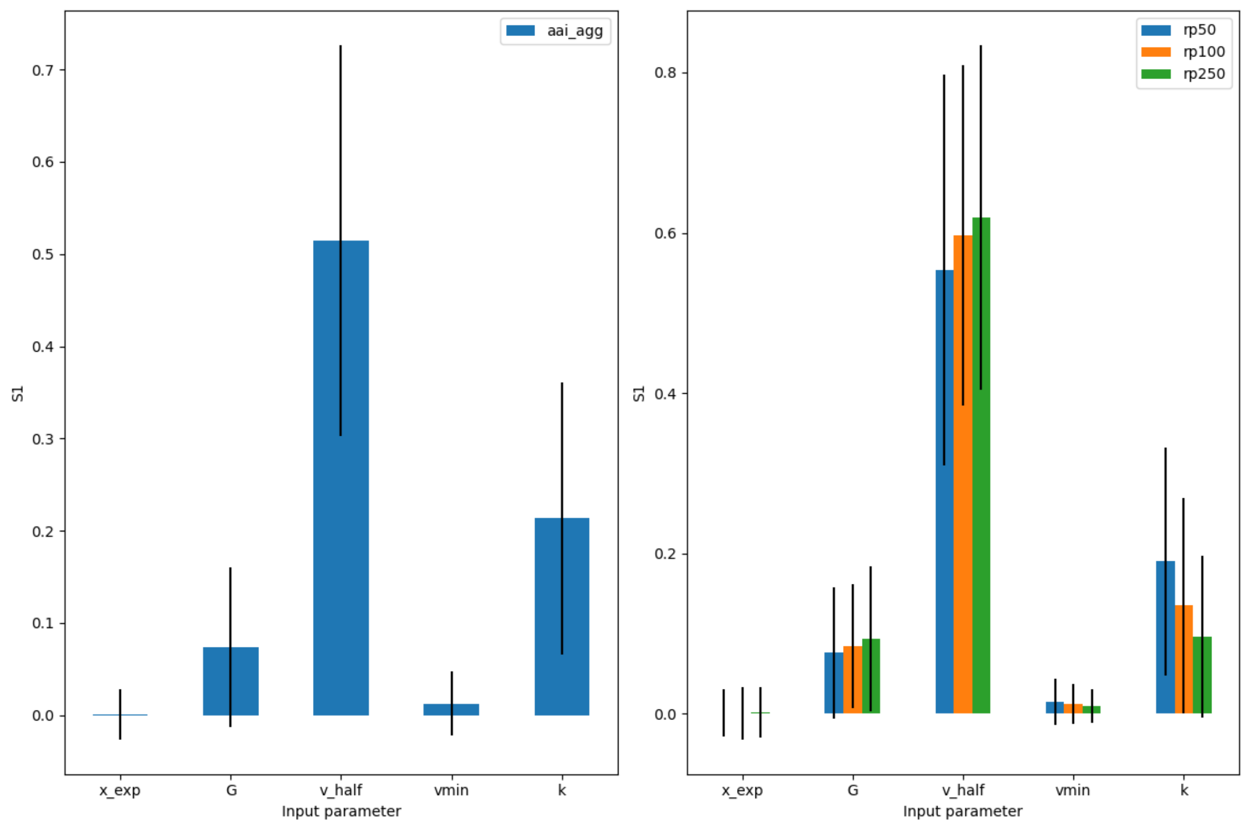
```
output_imp.get_largest_si(salib_si='S1')
```

```
    metric   param param2        si
0  aai_agg  v_half   None  0.514220
1     rp50  v_half   None  0.553640
2    rp100  v_half   None  0.596659
3    rp250  v_half   None  0.619366
```

The value of the sensitivity indices can be plotted for each metric that is one-dimensional (`eai_exp` and `at_event` are not shown in this plot).

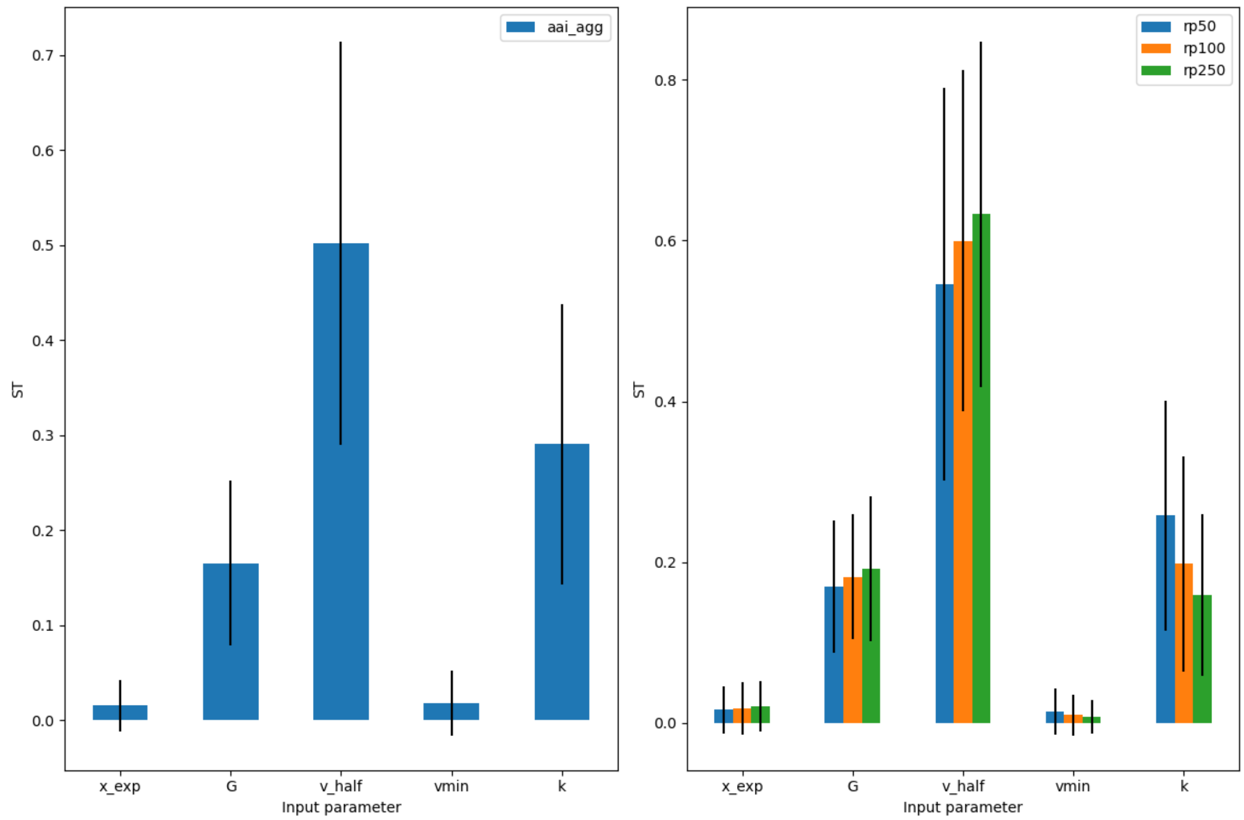
We see that both the errors in `freq_curve` and in `aai_agg` are mostly determined by `x_exp` and `v_half`. Finally, we see small differences in the sensitivity of the different return periods.

```
# Default for 'sobol' is to plot 'S1' sensitivity index.
output_imp.plot_sensitivity(figsize=(12,8));
```



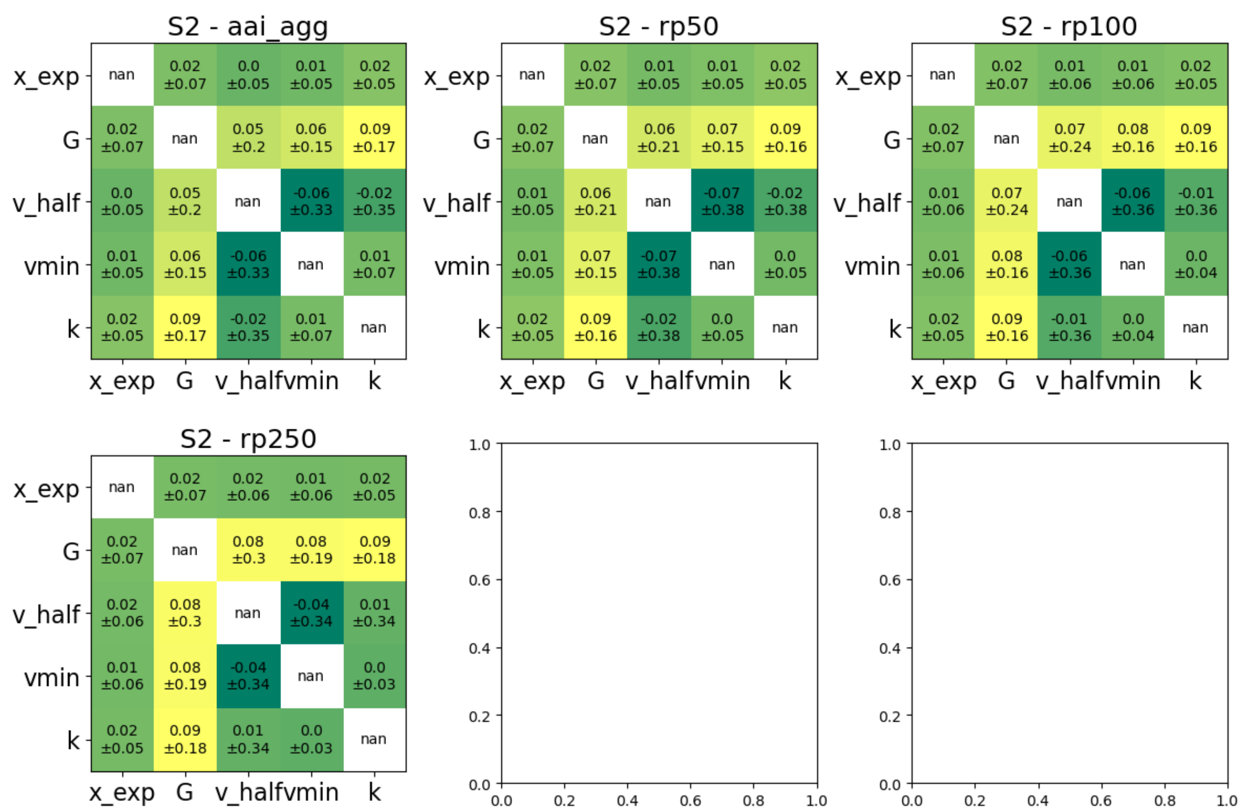
Note that since we have quite a few measures, the imp_meas_fut and imp_meas_pres plots are too crowded. We can select only the other metrics easily. In addition, instead of showing first order sensitivity 'S1', we can plot the total sensitivity 'ST'.

```
output_imp.plot_sensitivity(salib_si = 'ST', figsize=(12,8));
```



One can also vizualise the second-order sensitivity indices in the form of a correlation matrix.

```
output_imp.plot_sensitivity_second_order(figsize=(12,8));
```

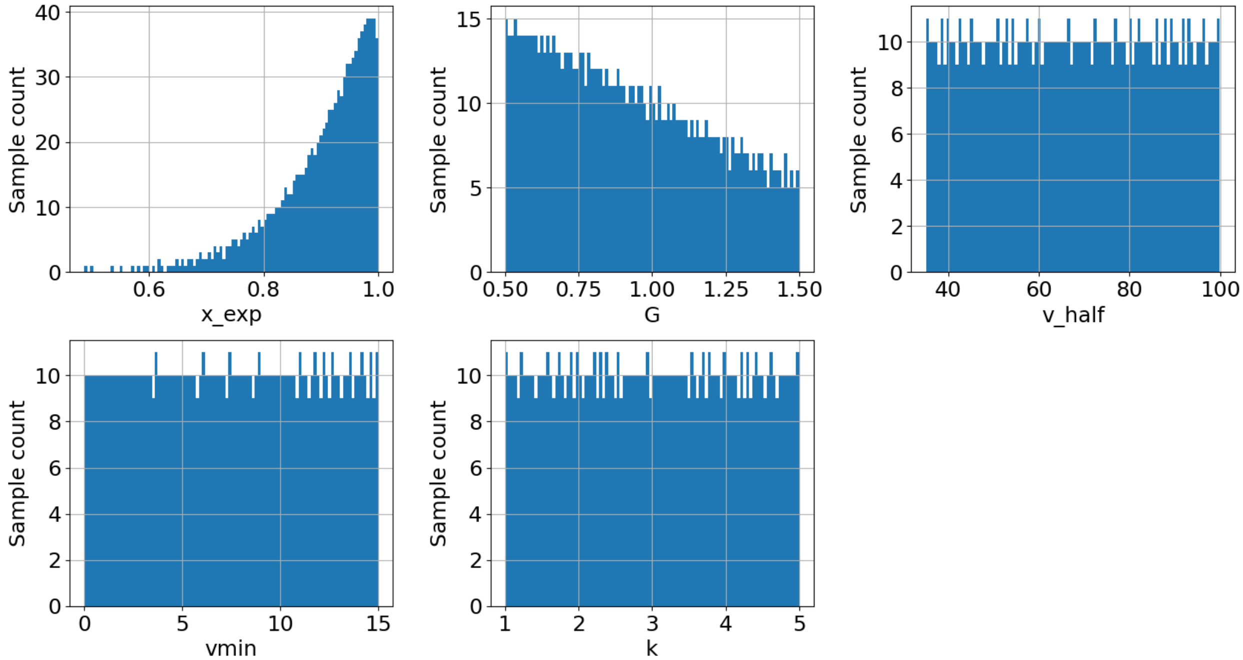


A few non-default parameters

We shall use the same uncertainty variables as in the previous section but show a few possibilities to use non-default method arguments.

```
# Sampling method "latin" hypercube instead of `saltelli`.
from climada.engine.unsequa import CalcImpact

calc_imp2 = CalcImpact(exp_iv, impf_iv, haz)
output_imp2 = calc_imp2.make_sample(N=1000, sampling_method='latin')
```

```
output_imp2.plot_sample(figsize=(15,8));
```




```
# Compute also the distribution of the metric `eai_exp`
# To speed-up the comutations, we can use more than one process
# Note that for large dataset a single process might be more efficient
import time

calc_imp2 = CalcImpact(exp_iv, impf_iv, haz)
output_imp2 = calc_imp2.make_sample(N=1000, sampling_method='latin')

start = time.time()
output_imp2 = calc_imp2.uncertainty(output_imp2, rp = [50, 100, 250], calc_eai_
↪exp=True, calc_at_event=True, processes=4)
end = time.time()
time_passed = end-start
print(f'Time passed with pool: {time_passed}')
```

```
Time passed with pool: 2.8349649906158447
```

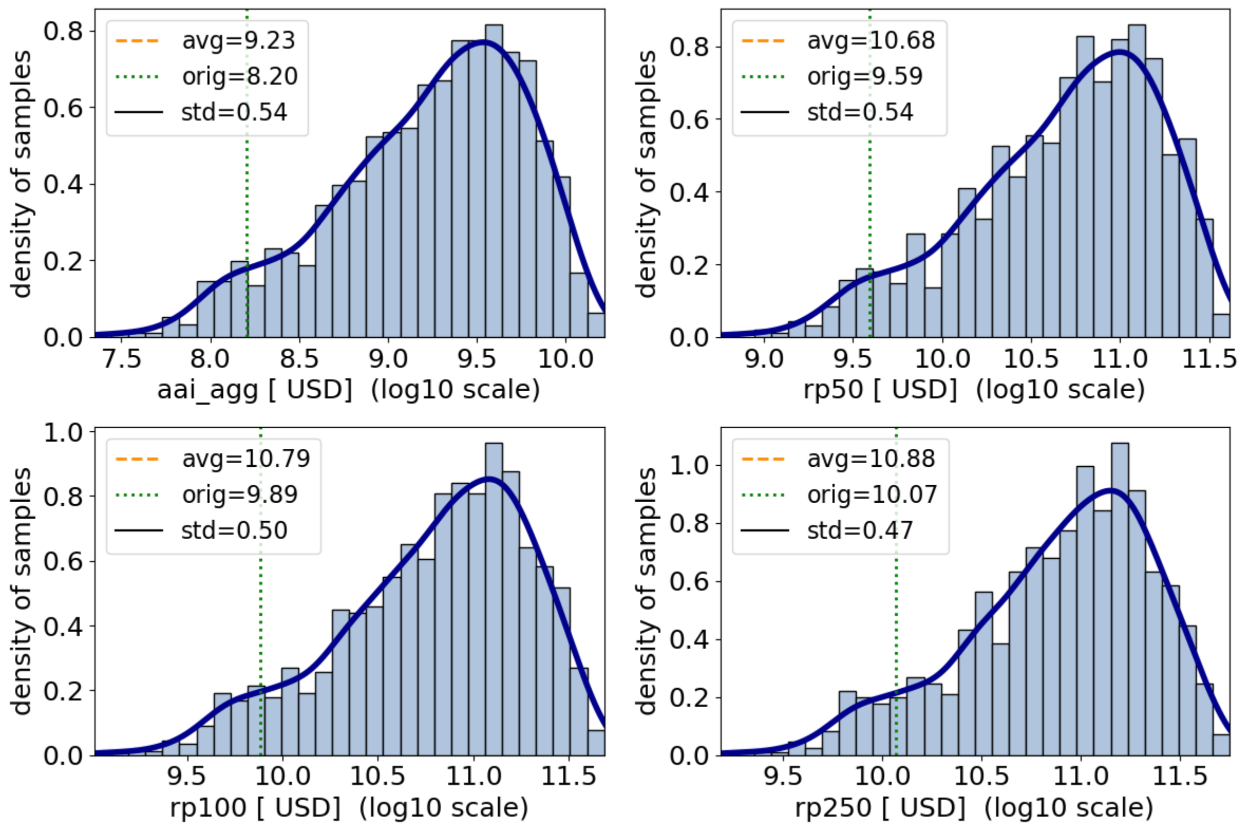
```
from climada.engine.unsequa import CalcImpact
import time
```

```
calc_imp2 = CalcImpact(exp_iv, impf_iv, haz)
output_imp2 = calc_imp2.make_sample(N=1000, sampling_method='latin')

start2 = time.time()
output_imp2 = calc_imp2.uncertainty(output_imp2, rp = [50, 100, 250], calc_eai_
↪exp=True, calc_at_event=True)
end2 = time.time()
time_passed_nopool = end2-start2
print(f'Time passed without pool: {time_passed_nopool}')
```

```
Time passed without pool: 8.287853956222534
```

```
# Add the original value of the impacts (without uncertainty) to the uncertainty plot
from climada.engine import ImpactCalc
imp = ImpactCalc(exp_base, impf_func(), haz).impact(assign_centroids=False)
aai_agg_o = imp.aai_agg
freq_curve_o = imp.calc_freq_curve([50, 100, 250]).impact
orig_list = [aai_agg_o] + list(freq_curve_o) +[1]
```

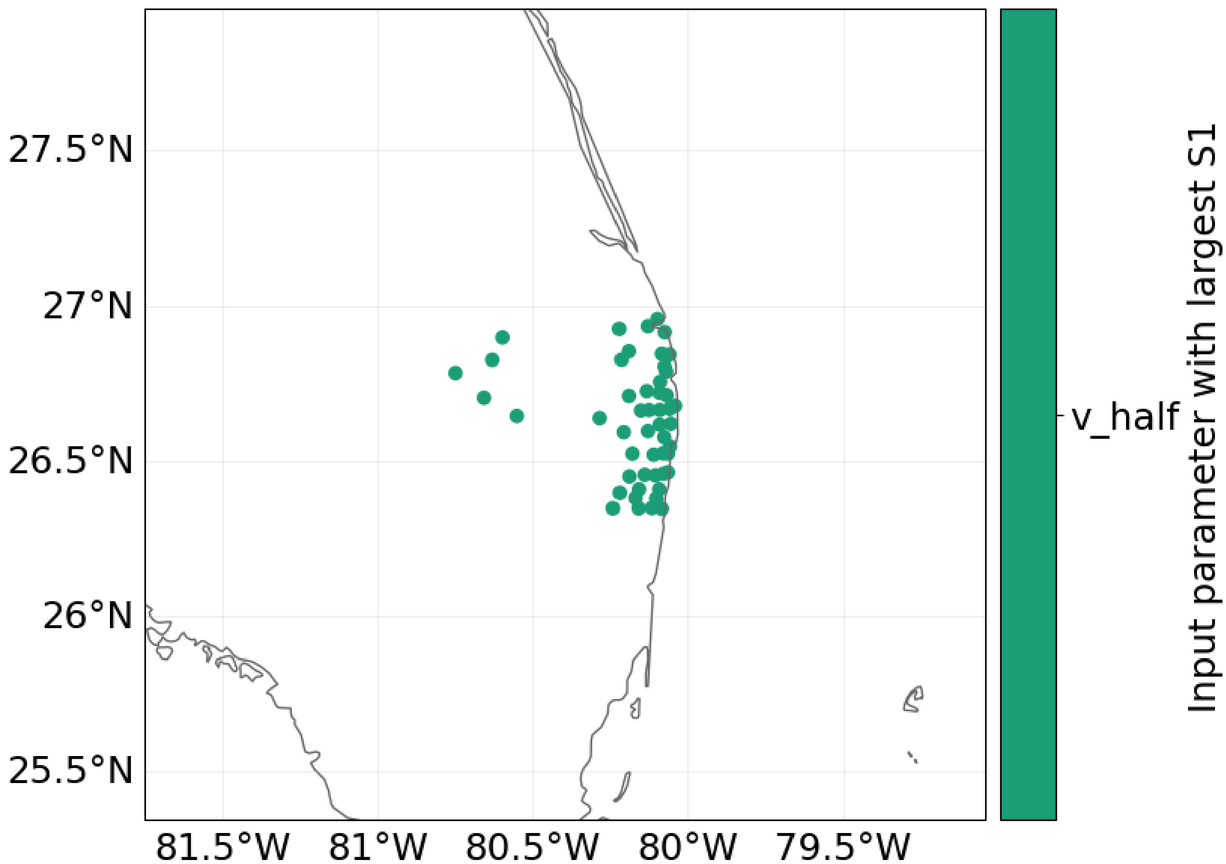
```
# plot the aai_agg and freq_curve uncertainty only
# use logarithmic x-scale
output_imp2.plot_uncertainty(metric_list=['aai_agg', 'freq_curve'], orig_list=orig_
↪list, log=True, figsize=(12,8));
```




```
# Use the method 'rbd_fast' which is recommend in pair with 'latin'. In addition,␣
↪change one of the kwargs
# (M=15) of the salib sampling method.
output_imp2 = calc_imp2.sensitivity(output_imp2, sensitivity_method='rbd_fast',␣
↪sensitivity_kwargs = {'M': 15})
```

Since we computed the distribution and sensitivity indices for the total impact at each exposure point, we can plot a map of the largest sensitivity index in each exposure location. For every location, the most sensitive parameter is `v_half`, meaning that the average annual impact at each location is most sensitivity to the ucnertainty in the impact function slope scaling parameter.

```
output_imp2.plot_sensitivity_map();
```




```
output_imp2.get_largest_si(salib_si='S1', metric_list=['eai_exp']).tail()
```

```
   metric   param param2        si
45     45  v_half   None  0.471587
46     46  v_half   None  0.471587
47     47  v_half   None  0.471587
48     48  v_half   None  0.467530
49     49  v_half   None  0.471587
```

11.2.6 CalcDeltaImpact

The main goal of this class is to perform an uncertainty and sensitivity analysis of the "delta" impact between a reference state and future (or any other "to be compared") state.

Classical example: risk increase in the future with climate change and socio economic development. In this case, the uncertainty and sensitivity analysis in performed on the estimated risk (delta) increase in the future relative to the present-day baseline.

The uncertainty and sensitivity analysis for CalcDeltaImpact is completely analogous to the Impact case. It is slightly more complex as there are more input variables.

Note, the logic of this class works with any comparison between an initial (reference and final (altered) risk or impact state and is not limited to the scope of climate change and socio-economic development in the future.

Set the Input Vars

We'll work through an analogous example as in CalcImpact next.

```
import numpy as np

from climada.entity import ImpactFunc, ImpactFuncSet, Exposures
from climada.util.constants import EXP_DEMO_H5, HAZ_DEMO_H5
from climada.hazard import Centroids, TCTracks, Hazard, TropCyclone

def impf_func(G=1, v_half=84.7, vmin=25.7, k=3, _id=1):

    def xhi(v, v_half, vmin):
        return max([(v - vmin), 0]) / (v_half - vmin)

    def sigmoid_func(v, G, v_half, vmin, k):
        return G * xhi(v, v_half, vmin)**k / (1 + xhi(v, v_half, vmin)**k)

    #In-function imports needed only for parallel computing on Windows
    intensity_unit = 'm/s'
    intensity = np.linspace(0, 150, num=100)
    mdd = np.repeat(1, len(intensity))
    paa = np.array([sigmoid_func(v, G, v_half, vmin, k) for v in intensity])
    imp_fun = ImpactFunc("TC", _id, intensity, mdd, paa, intensity_unit)
    imp_fun.check()
    impf_set = ImpactFuncSet([imp_fun])
    return impf_set
```

Load the hazard set and apply climate change factors to it. This yields a hazard representation in 2050 under 4 RCP scenarios. For a full documentation of this function please refer to the *TropCyclone* tutorial.

```
# load historical hazard set
haz = TropCyclone.from_hdf5(HAZ_DEMO_H5)
haz.basin = ["NA"] * haz.size

# apply climate change factors
haz_26 = haz.apply_climate_scenario_knu(ref_year=2050, rcp_scenario=26)
haz_45 = haz.apply_climate_scenario_knu(ref_year=2050, rcp_scenario=45)
haz_60 = haz.apply_climate_scenario_knu(ref_year=2050, rcp_scenario=60)
haz_85 = haz.apply_climate_scenario_knu(ref_year=2050, rcp_scenario=85)

# pack future hazard sets into dictionary - we want to sample from this dictionary␣
↪later
haz_fut_list = [haz_26, haz_45, haz_60, haz_85]
tc_haz_fut_dict = {}
for r, rcp in enumerate(['26', '45', '60', '85']):
    tc_haz_fut_dict[rcp] = haz_fut_list[r]
```

```
exp_base = Exposures.from_hdf5(EXP_DEMO_H5)
#It is a good idea to assign the centroids to the base exposures in order to avoid␣
↪repeating this
# potentially costly operation for each sample.
exp_base.assign_centroids(haz)
def exp_base_func(x_exp, exp_base):
    exp = exp_base.copy()
    exp.gdf.value *= x_exp
    return exp
```

(continues on next page)

(continued from previous page)

```
from functools import partial
exp_func = partial(exp_base_func, exp_base=exp_base)
```

```
import scipy as sp
from climada.engine.unsequa import InputVar

exp_distr = {"x_exp": sp.stats.beta(10, 1.1)} #This is not really a reasonable␣
↪distribution but is used
                                              #here to show that you can use any␣
↪scipy distribution.

exp_iv = InputVar(exp_func, exp_distr)

impf_distr = {
    "G": sp.stats.truncnorm(0.5, 1.5),
    "v_half": sp.stats.uniform(35, 65),
    "vmin": sp.stats.uniform(0, 15),
    "k": sp.stats.uniform(1, 4)
    }
impf_iv = InputVar(impf_func, impf_distr)
```

Next we define the function for the future hazard representation. It's a simple function that allows us to draw from the hazard dictionary of hazard sets under different RCP scenarios. Note, we do not investigate other hazard related uncertainties in this example.

```
rcp_key = {0: '26',
           1: '45',
           2: '60',
           3: '85'}

# future
def haz_fut_func(rcp_scenario):
    haz_fut = tc_haz_fut_dict[rcp_key[rcp_scenario]]
    return haz_fut

haz_fut_distr = {"rcp_scenario": sp.stats.randint(0, 4)}

haz_fut_iv = InputVar(haz_fut_func, haz_fut_distr)
```

Compute uncertainty and sensitivity

In contrast to CalcImpact, we define InputVars for initial and final states of exposure, impact function, hazard. This class requires 6 input variables. For the sake of simplicity, we did not define varying input variables for the initial and future exposure and vulernability in the example. Hence, the exp_iv and impf_iv are passed to CalcDeltaImpact twice.

```
from climada.engine.unsequa import CalcDeltaImpact
calc_imp = CalcDeltaImpact(exp_iv, impf_iv, haz,
                           exp_iv, impf_iv, haz_fut_iv)
```

```
2024-01-25 15:36:53,385 - climada.engine.unsequa.calc_base - WARNING -

The input parameter x_exp is shared among at least 2 input variables. Their␣
```

(continues on next page)

(continued from previous page)

```
↪uncertainty is thus computed with the same samples for this input paramter.


2024-01-25 15:36:53,389 - climada.engine.unsequa.calc_base - WARNING -

The input parameter G is shared among at least 2 input variables. Their uncertainty␣
↪is thus computed with the same samples for this input paramter.


2024-01-25 15:36:53,390 - climada.engine.unsequa.calc_base - WARNING -

The input parameter v_half is shared among at least 2 input variables. Their␣
↪uncertainty is thus computed with the same samples for this input paramter.


2024-01-25 15:36:53,393 - climada.engine.unsequa.calc_base - WARNING -

The input parameter vmin is shared among at least 2 input variables. Their␣
↪uncertainty is thus computed with the same samples for this input paramter.


2024-01-25 15:36:53,394 - climada.engine.unsequa.calc_base - WARNING -

The input parameter k is shared among at least 2 input variables. Their uncertainty␣
↪is thus computed with the same samples for this input paramter.
```

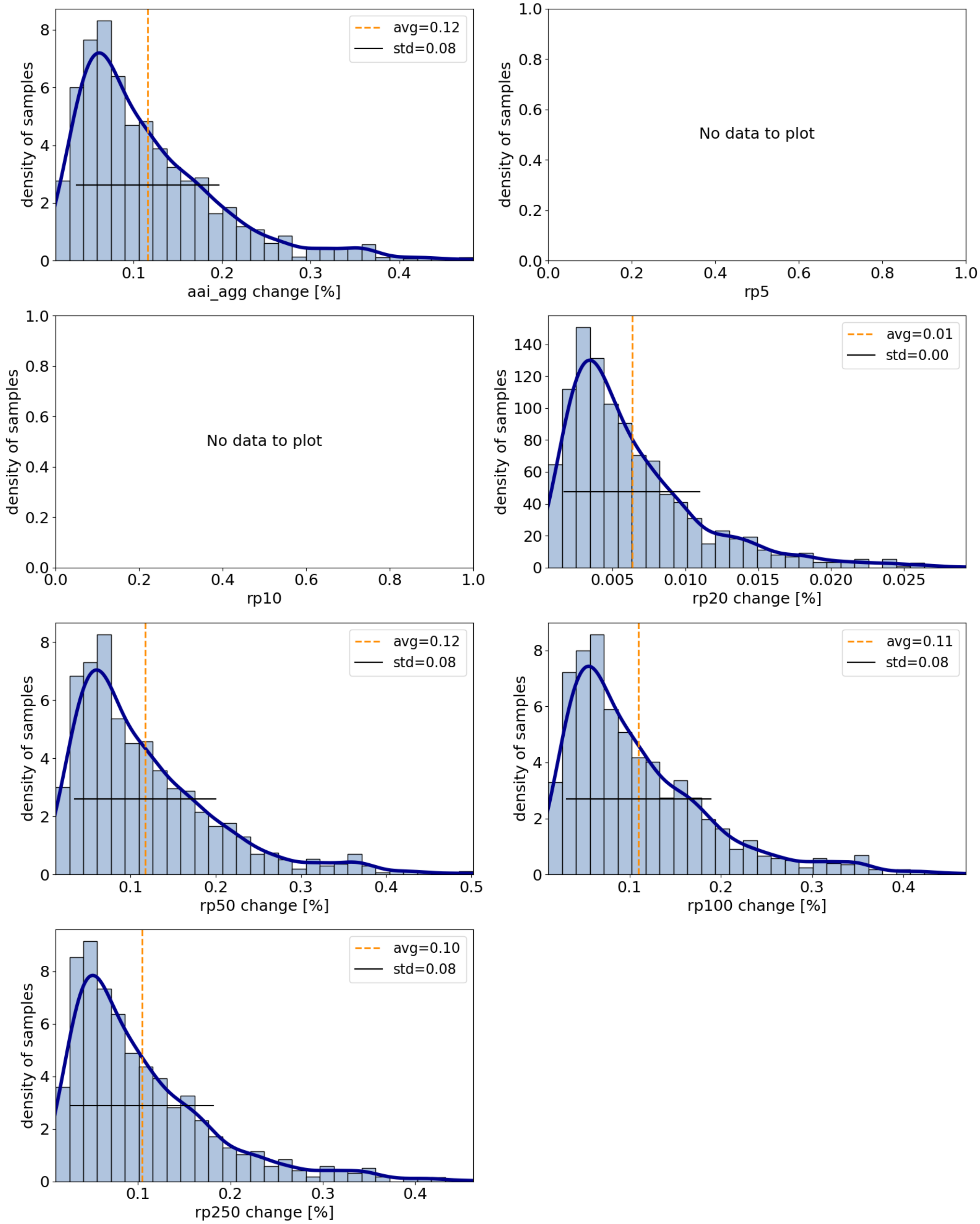
```
output_imp = calc_imp.make_sample(N=2**7)
output_imp.get_samples_df().tail()

output_imp = calc_imp.uncertainty(output_imp)
```

Plotting functionalities work analogous to CalcImpact. By setting `calc_delta=True`, the axis labels are adjusted.

```
from climada.engine.unsequa import UncOutput
output_imp.plot_uncertainty(calc_delta=True)
```

```
No data to plot for 'rp5'.
No data to plot for 'rp10'.
```

```
array([[<Axes: xlabel='aai_agg change [%]', ylabel='density of samples'>,
        <Axes: xlabel='rp5', ylabel='density of samples'>],
       [<Axes: xlabel='rp10', ylabel='density of samples'>,
        <Axes: xlabel='rp20 change [%]', ylabel='density of samples'>],
       [<Axes: xlabel='rp50 change [%]', ylabel='density of samples'>,
        <Axes: xlabel='rp100 change [%]', ylabel='density of samples'>],
       [<Axes: xlabel='rp250 change [%]', ylabel='density of samples'>,
        <Axes: >]], dtype=object)
```

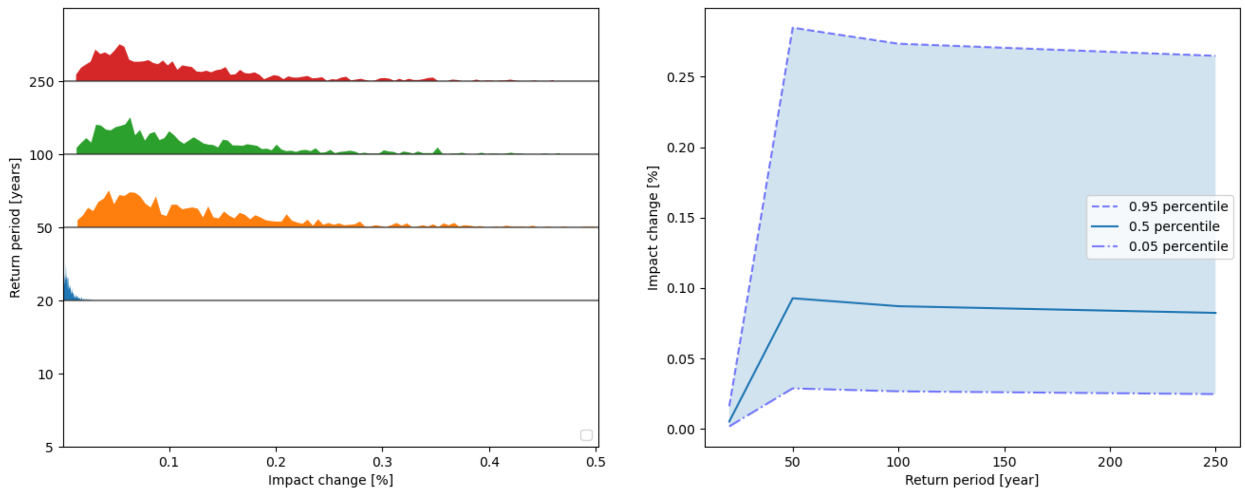



```
from climada.engine.unsequa import UncOutput
output_imp.plot_rp_uncertainty(calc_delta=True)
```

```
No artists with labels found to put in legend.  Note that artists whose label start␣
↪with an underscore are ignored when legend() is called with no argument.
```

```
Skipping plot for 'rp5': insufficient data.
Skipping plot for 'rp10': insufficient data.
```

```
array([<Axes: xlabel='Impact change [%]', ylabel='Return period [years]'>,
       <Axes: xlabel='Return period [year]', ylabel='Impact change [%]'>],
      dtype=object)
```

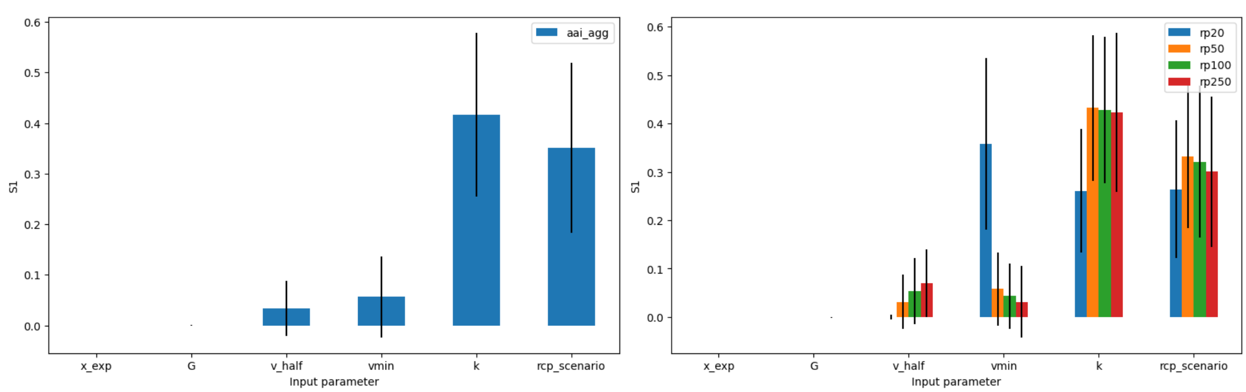



```
# compute sensitivity
output_imp = calc_imp.sensitivity(output_imp)
```

```
# plot sensitivity
output_imp.plot_sensitivity()
```

```
2024-01-25 15:37:27,753 - climada.engine.unsequa.unc_output - WARNING - All-NaN␣
↪columns encountered: ['rp5', 'rp10']
```

```
array([<Axes: xlabel='Input parameter', ylabel='S1'>,
       <Axes: xlabel='Input parameter', ylabel='S1'>], dtype=object)
```



The rest of the functionalities that apply to CalcImpact also work for the CalcDeltaImpact class. Hence, refer to the sections above for details.

11.2.7 CalcCostBenefit

The uncertainty and sensitivity analysis for CostBenefit is completely analogous to the Impact case. It is slightly more complex as there are more input variables.

Set the Input Vars

```
import copy
from climada.util.constants import ENT_DEMO_TODAY, ENT_DEMO_FUTURE, HAZ_DEMO_H5
from climada.entity import Entity
from climada.hazard import Hazard

# Entity today has an uncertainty in the total asset value
def ent_today_func(x_ent):
    #In-function imports needed only for parallel computing on Windows
    from climada.entity import Entity
    from climada.util.constants import ENT_DEMO_TODAY
    entity = Entity.from_excel(ENT_DEMO_TODAY)
    entity.exposures.ref_year = 2018
    entity.exposures.gdf.value *= x_ent
    return entity

# Entity in the future has a +- 10% uncertainty in the cost of all the adapatation␣
↪measures
def ent_fut_func(m_fut_cost):
    #In-function imports needed only for parallel computing on Windows
    from climada.entity import Entity
    from climada.util.constants import ENT_DEMO_FUTURE
    entity = Entity.from_excel(ENT_DEMO_FUTURE)
    entity.exposures.ref_year = 2040
    for meas in entity.measures.get_measure('TC'):
        meas.cost *= m_fut_cost
    return entity

haz_base = Hazard.from_hdf5(HAZ_DEMO_H5)
# The hazard intensity in the future is also uncertainty by a multiplicative factor
def haz_fut(x_haz_fut, haz_base):
    #In-function imports needed only for parallel computing on Windows
    import copy
    from climada.hazard import Hazard
    from climada.util.constants import HAZ_DEMO_H5
    haz = copy.deepcopy(haz_base)
    haz.intensity = haz.intensity.multiply(x_haz_fut)
    return haz
from functools import partial
haz_fut_func = partial(haz_fut, haz_base=haz_base)
```

Check that costs for measures are changed as desired.

```
costs_1 = [meas.cost for meas in ent_fut_func(1).measures.get_measure('TC')]
costs_05 = [meas.cost for meas in ent_fut_func(0.5).measures.get_measure('TC')]
```

(continues on next page)

(continued from previous page)

```
print(f"\nThe cost for m_fut_cost=1 are {costs_1}\n"
     f"The cost for m_fut_cost=0.5 are {costs_05}");
```

```
The cost for m_fut_cost=1 are [1311768360.8515418, 1728000000.0, 8878779433.630093,␣
↪9200000000.0]
The cost for m_fut_cost=0.5 are [655884180.4257709, 864000000.0, 4439389716.815046,␣
↪4600000000.0]
```

Define the InputVars

```
import scipy as sp
from climada.engine.unsequa import InputVar

haz_today = haz_base

haz_fut_distr = {"x_haz_fut": sp.stats.uniform(1, 3),
              }
haz_fut_iv = InputVar(haz_fut_func, haz_fut_distr)

ent_today_distr = {"x_ent":  sp.stats.uniform(0.7, 1)}
ent_today_iv = InputVar(ent_today_func, ent_today_distr)

ent_fut_distr = {"m_fut_cost":  sp.stats.norm(1, 0.1)}
ent_fut_iv = InputVar(ent_fut_func, ent_fut_distr)
```

```
ent_avg = ent_today_iv.evaluate()
ent_avg.exposures.gdf.head()
```

```
    latitude  longitude         value  deductible         cover  impf_TC  \
0  26.933899 -80.128799  1.671301e+10           0  1.392750e+10        1
1  26.957203 -80.098284  1.511528e+10           0  1.259606e+10        1
2  26.783846 -80.748947  1.511528e+10           0  1.259606e+10        1
3  26.645524 -80.550704  1.511528e+10           0  1.259606e+10        1
4  26.897796 -80.596929  1.511528e+10           0  1.259606e+10        1

     Value_2010
0  5.139301e+09
1  4.647994e+09
2  4.647994e+09
3  4.647994e+09
4  4.647994e+09
```

Compute cost benefit uncertainty and sensitivity using default methods

For examples of how to use non-defaults please see the *impact example*

```
from climada.engine.unsequa import CalcCostBenefit

unc_cb = CalcCostBenefit(haz_input_var=haz_today, ent_input_var=ent_today_iv,
                     haz_fut_input_var=haz_fut_iv, ent_fut_input_var=ent_fut_iv)
```

```
output_cb= unc_cb.make_sample(N=10, sampling_kwargs={'calc_second_order':False})
output_cb.get_samples_df().tail()
```

```
      x_ent  x_haz_fut  m_fut_cost
45  1.35625    2.96875    0.813727
46  1.04375    2.96875    0.813727
47  1.35625    2.03125    0.813727
48  1.35625    2.96875    0.899001
49  1.04375    2.03125    0.899001
```

For longer computations, it is possible to use a pool for parallel computation.

```
#without pool
output_cb = unc_cb.uncertainty(output_cb)

#with pool
output_cb = unc_cb.uncertainty(output_cb, processes=4)
```

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148           13.8606        10.6498
Beach nourishment          1.71445           10.7904         6.29377
Seawall                    8.80916            0.175596       0.0199334
Building code              9.12786           29.4038         3.22132

--------------------  --------  --------
Total climate risk:   117.615   (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3848  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148           13.8606        10.6498
Beach nourishment          1.71445           10.7904         6.29377
Seawall                    8.80916            0.175596       0.0199334
Building code              9.12786           29.4038         3.22132

--------------------  --------  --------
Total climate risk:   117.615   (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3848  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148           14.0781        10.817
Beach nourishment          1.71445           10.968          6.39739
Seawall                    8.80916            0.175596       0.0199334
Building code              9.12786           29.5124         3.23322

--------------------  --------  --------
Total climate risk:   118.05    (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3155  (USD bn)
```

(continues on next page)

(continued from previous page)

```
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148          14.1012        10.8347
Beach nourishment          1.71445          10.9632         6.39461
Seawall                    8.80916           0.0376243      0.00427104
Building code              9.12786          13.3845         1.46633

--------------------  --------  --------
Total climate risk:   53.5379   (USD bn)
Average annual risk:   6.15933  (USD bn)
Residual risk:        15.0513   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612           13.8606         8.90716
Beach nourishment          2.04988           10.7904         5.2639
Seawall                   10.5327             0.175596       0.0166716
Building code             10.9137            29.4038         2.69421

--------------------  --------  --------
Total climate risk:   117.615   (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3848  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612          14.3188         9.20163
Beach nourishment          2.04988          11.1409         5.4349
Seawall                   10.5327            0.0376243      0.00357216
Building code             10.9137           13.4931         1.23634

--------------------  --------  --------
Total climate risk:   53.9724   (USD bn)
Average annual risk:   6.15933  (USD bn)
Residual risk:        14.982    (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612             6.59956       4.24104
Beach nourishment          2.04988             5.16368       2.51902
Seawall                   10.5327              3.55475       0.337498
Building code             10.9137             48.016         4.3996

--------------------  --------  --------
Total climate risk:   192.064   (USD bn)
Average annual risk:   22.2359  (USD bn)
Residual risk:        128.73    (USD bn)
--------------------  --------  --------
```

(continues on next page)

(continued from previous page)

```
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612             6.43034       4.1323
Beach nourishment          2.04988             5.02552       2.45161
Seawall                   10.5327              3.55475       0.337498
Building code             10.9137             47.9315        4.39186

--------------------  --------  --------
Total climate risk:   191.726   (USD bn)
Average annual risk:   22.2359  (USD bn)
Residual risk:        128.784   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612             7.59067         4.87796
Beach nourishment          2.04988             5.96389         2.90939
Seawall                   10.5327              1.31269         0.12463
Building code             10.9137             43.2513          3.96302

--------------------  --------  --------
Total climate risk:   173.005   (USD bn)
Average annual risk:   20.0179  (USD bn)
Residual risk:        114.887   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148             6.59956       5.0708
Beach nourishment          1.71445             5.16368       3.01186
Seawall                    8.80916             3.55475       0.403528
Building code              9.12786            48.016         5.26038

--------------------  --------  --------
Total climate risk:   192.064   (USD bn)
Average annual risk:   22.2359  (USD bn)
Residual risk:        128.73    (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148             7.42146       5.70231
Beach nourishment          1.71445             5.82573       3.39801
Seawall                    8.80916             1.31269       0.149014
Building code              9.12786            43.1668        4.72913

--------------------  --------  --------
Total climate risk:   172.667   (USD bn)
Average annual risk:   20.0179  (USD bn)
Residual risk:        114.941   (USD bn)
--------------------  --------  --------
Net Present Values
```

(continues on next page)

(continued from previous page)

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992         10.5404        8.71168
Beach nourishment          1.59383          8.59532       5.39285
Seawall                    8.18941          0.0184996     0.00225897
Building code              8.48569          7.53759       0.88827

--------------------  --------  --------
Total climate risk:   30.1504   (USD bn)
Average annual risk:   3.37008  (USD bn)
Residual risk:         3.45852  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992         10.5646        8.73166
Beach nourishment          1.59383          8.61505       5.40524
Seawall                    8.18941          0.0184996     0.00225897
Building code              8.48569          7.54966       0.889693

--------------------  --------  --------
Total climate risk:   30.1986   (USD bn)
Average annual risk:   3.37008  (USD bn)
Residual risk:         3.45082  (USD bn)
--------------------  --------  --------
Net Present Values
```

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992          12.8863       10.6505
Beach nourishment          1.59383           9.98362       6.2639
Seawall                    8.18941           0.257712      0.0314689
Building code              8.48569          33.7244        3.97427

--------------------  --------  --------
Total climate risk:   134.898   (USD bn)
Average annual risk:   15.5605  (USD bn)
Residual risk:         78.0457  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992         10.5404        8.71168
Beach nourishment          1.59383          8.59532       5.39285
Seawall                    8.18941          0.0184996     0.00225897
Building code              8.48569          7.53759       0.88827

--------------------  --------  --------
Total climate risk:   30.1504   (USD bn)
Average annual risk:   3.37008  (USD bn)
Residual risk:         3.45852  (USD bn)
--------------------  --------  --------
Net Present Values
```

(continues on next page)

(continued from previous page)

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992           12.9104       10.6705
Beach nourishment          1.59383           10.0034        6.27629
Seawall                    8.18941            0.257712      0.0314689
Building code              8.48569           33.7365        3.97569

--------------------  --------  --------
Total climate risk:   134.946   (USD bn)
Average annual risk:   15.5605  (USD bn)
Residual risk:         78.038   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774            8.44955       6.08873
Beach nourishment          1.82807            6.4567        3.53197
Seawall                    9.39298            0.895618      0.0953497
Building code              9.7328            41.407         4.25438

--------------------  --------  --------
Total climate risk:   165.628   (USD bn)
Average annual risk:   19.1818  (USD bn)
Residual risk:        108.419   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774            8.47373       6.10615
Beach nourishment          1.82807            6.47644       3.54276
Seawall                    9.39298            0.895618      0.0953497
Building code              9.7328            41.4191        4.25562

--------------------  --------  --------
Total climate risk:   165.676   (USD bn)
Average annual risk:   19.1818  (USD bn)
Residual risk:        108.411   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774          1.39186         1.00297
Beach nourishment          1.82807          1.13491         0.62082
Seawall                    9.39298          0.000227424     2.42121e-05
Building code              9.7328           0.76062         0.0781501

--------------------  ---------  --------
Total climate risk:    3.04248   (USD bn)
Average annual risk:   0.260244  (USD bn)
Residual risk:        -0.245137  (USD bn)
--------------------  ---------  --------
Net Present Values
```

(continues on next page)

(continued from previous page)

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.38774           8.44955         6.08873
Beach nourishment         1.82807           6.4567          3.53197
Seawall                   9.39298           0.895618        0.0953497
Building code             9.7328           41.407           4.25438

--------------------  --------  --------
Total climate risk:   165.628   (USD bn)
Average annual risk:   19.1818  (USD bn)
Residual risk:        108.419   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.38774         1.41604         1.02039
Beach nourishment         1.82807         1.15464         0.631618
Seawall                   9.39298         0.000227424     2.42121e-05
Building code             9.7328          0.77269         0.0793903

--------------------  ---------  --------
Total climate risk:    3.09076   (USD bn)
Average annual risk:   0.260244  (USD bn)
Residual risk:        -0.252837  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.44426         2.77518         1.92153
Beach nourishment         1.90253         2.24443         1.17971
Seawall                   9.77553         0.00264385      0.000270456
Building code            10.1292          1.68273         0.166127

--------------------  ---------  --------
Total climate risk:   6.73092    (USD bn)
Average annual risk:  0.678263   (USD bn)
Residual risk:        0.0259328  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.44426         2.70266         1.87132
Beach nourishment         1.90253         2.18521         1.14859
Seawall                   9.77553         0.00264385      0.000270456
Building code            10.1292          1.64652         0.162552

--------------------  ---------  --------
Total climate risk:   6.58607    (USD bn)
Average annual risk:  0.678263   (USD bn)
Residual risk:        0.0490347  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
```

(continues on next page)

(continued from previous page)

```
-----------------  ---------------  ------------------  --------------
Mangroves                 1.44426         5.43516        3.76329
Beach nourishment         1.90253         4.37024        2.29707
Seawall                   9.77553         0.00764219     0.000781768
Building code            10.1292          3.61055        0.35645

--------------------  --------  --------
Total climate risk:   14.4422   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0186   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.44426         2.77518        1.92153
Beach nourishment         1.90253         2.24443        1.17971
Seawall                   9.77553         0.00264385     0.000270456
Building code            10.1292          1.68273        0.166127

--------------------  ---------  --------
Total climate risk:   6.73092    (USD bn)
Average annual risk:  0.678263   (USD bn)
Residual risk:        0.0259328  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.44426         5.36264        3.71308
Beach nourishment         1.90253         4.31103        2.26595
Seawall                   9.77553         0.00764219     0.000781768
Building code            10.1292          3.57434        0.352875

--------------------  --------  --------
Total climate risk:   14.2973   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0417   (USD bn)
--------------------  --------  --------
Net Present Values
```

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.259           10.4834          8.32677
Beach nourishment         1.65849          8.23954         4.96809
Seawall                   8.52163          0.415328        0.0487381
Building code             8.82993         36.8908          4.17793

--------------------  --------  --------
Total climate risk:   147.563   (USD bn)
Average annual risk:   17.0232  (USD bn)
Residual risk:         91.5342  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
```

(continues on next page)

(continued from previous page)

```
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259             10.4109        8.26917
Beach nourishment          1.65849            8.18032       4.93239
Seawall                    8.52163            0.415328      0.0487381
Building code              8.82993           36.8546        4.17383

--------------------  --------  --------
Total climate risk:   147.418   (USD bn)
Average annual risk:   17.0232  (USD bn)
Residual risk:         91.5573  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259              8.79133       6.98278
Beach nourishment          1.65849            6.82168       4.11319
Seawall                    8.52163            0.621468      0.0729283
Building code              8.82993           39.3223        4.45329

--------------------  --------  --------
Total climate risk:   157.289   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.732   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259             10.4834        8.32677
Beach nourishment          1.65849            8.23954       4.96809
Seawall                    8.52163            0.415328      0.0487381
Building code              8.82993           36.8908        4.17793

--------------------  --------  --------
Total climate risk:   147.563   (USD bn)
Average annual risk:   17.0232  (USD bn)
Residual risk:         91.5342  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259              8.71881       6.92518
Beach nourishment          1.65849            6.76247       4.07748
Seawall                    8.52163            0.621468      0.0729283
Building code              8.82993           39.286         4.44919

--------------------  --------  --------
Total climate risk:   157.144   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.755   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
```

(continues on next page)

(continued from previous page)

```
Mangroves                 1.34288        14.575        10.8535
Beach nourishment         1.76899        11.5436        6.52553
Seawall                   9.08939         0.0620626     0.00682803
Building code             9.41823        19.0306        2.02062

--------------------  --------  --------
Total climate risk:   76.1225   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9112   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.34288        14.3091       10.6555
Beach nourishment         1.76899        11.3265        6.40279
Seawall                   9.08939         0.0620626     0.00682803
Building code             9.41823        18.8979        2.00652

--------------------  --------  --------
Total climate risk:   75.5914   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9959   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.34288            7.41063        5.51844
Beach nourishment         1.76899            5.87997        3.32392
Seawall                   9.08939            2.58858        0.284791
Building code             9.41823           46.6706         4.95535

--------------------  --------  --------
Total climate risk:   186.682   (USD bn)
Average annual risk:   21.5984  (USD bn)
Residual risk:        124.133   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.13888        14.575        12.7977
Beach nourishment         1.50025        11.5436        7.69445
Seawall                   7.70855         0.0620626     0.00805114
Building code             7.98743        19.0306        2.38257

--------------------  --------  --------
Total climate risk:   76.1225   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9112   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.13888            7.14472        6.27348
```

(continues on next page)

(continued from previous page)

```
Beach nourishment         1.50025            5.66285        3.77461
Seawall                   7.70855            2.58858        0.335806
Building code             7.98743           46.5378         5.82638

--------------------  --------  --------
Total climate risk:   186.151   (USD bn)
Average annual risk:   21.5984  (USD bn)
Residual risk:        124.217   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.13888            7.63243        6.70172
Beach nourishment         1.50025            5.87772        3.91783
Seawall                   7.70855            1.82863        0.237221
Building code             7.98743           44.9771         5.63099

--------------------  -------  --------
Total climate risk:   179.909  (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.593  (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.13888            7.7533         6.80785
Beach nourishment         1.50025            5.97641        3.98361
Seawall                   7.70855            1.82863        0.237221
Building code             7.98743           45.0375         5.63855

--------------------  -------  --------
Total climate risk:   180.15   (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.554  (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.13888          14.9037        13.0863
Beach nourishment         1.50025          11.752          7.83335
Seawall                   7.70855           0.108627       0.0140917
Building code             7.98743          24.3925         3.05386

--------------------  -------  --------
Total climate risk:   97.5699  (USD bn)
Average annual risk:  11.2725  (USD bn)
Residual risk:        46.4132  (USD bn)
--------------------  -------  --------
Net Present Values
```

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.34288            7.63243        5.68361
```

(continues on next page)

(continued from previous page)

```
Beach nourishment          1.76899            5.87772        3.32264
Seawall                    9.08939            1.82863        0.201183
Building code              9.41823           44.9771         4.77554

--------------------  -------  --------
Total climate risk:   179.909  (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.593  (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288           15.0245        11.1883
Beach nourishment          1.76899           11.8507         6.69911
Seawall                    9.08939            0.108627       0.0119509
Building code              9.41823           24.4528         2.59633

--------------------  -------  --------
Total climate risk:   97.8114  (USD bn)
Average annual risk:  11.2725  (USD bn)
Residual risk:        46.3747  (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.32205          5.31429        4.01972
Beach nourishment          1.74155          4.27155        2.45273
Seawall                    8.9484           0.00764219     0.000854029
Building code              9.27214          3.5502         0.382889

--------------------  --------  --------
Total climate risk:   14.2008   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0571   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.32205          5.5802         4.22086
Beach nourishment          1.74155          4.48867        2.5774
Seawall                    8.9484           0.00764219     0.000854029
Building code              9.27214          3.68297        0.397208

--------------------  ---------  --------
Total climate risk:   14.7319    (USD bn)
Average annual risk:   1.57569   (USD bn)
Residual risk:         0.972395  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.32205            7.60825        5.75487
Beach nourishment          1.74155            5.85798        3.36366
```

(continues on next page)

(continued from previous page)

```
Seawall                    8.9484             1.82863       0.204353
Building code              9.27214           44.9651        4.84948

--------------------  -------  --------
Total climate risk:   179.86   (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.6    (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.36453          5.31429        3.89458
Beach nourishment          1.79751          4.27155        2.37637
Seawall                    9.23593          0.00764219     0.000827442
Building code              9.57007          3.5502         0.370969

--------------------  --------  --------
Total climate risk:   14.2008   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0571   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.36453             7.87416       5.77059
Beach nourishment          1.79751             6.0751        3.37973
Seawall                    9.23593             1.82863       0.197991
Building code              9.57007            45.0978        4.71238

--------------------  -------  --------
Total climate risk:   180.391  (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.516  (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.06742            8.67047       8.12282
Beach nourishment          1.40612            6.72299       4.78124
Seawall                    7.2249             0.621468      0.0860176
Building code              7.48629           39.2619        5.24451

--------------------  --------  --------
Total climate risk:   157.048   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.771   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.06742            8.5496        8.00959
Beach nourishment          1.40612            6.6243        4.71105
Seawall                    7.2249             0.621468      0.0860176
```

(continues on next page)

(continued from previous page)

```
Building code             7.48629          39.2016        5.23645

--------------------  --------  --------
Total climate risk:   156.806   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.809   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.06742          14.5509        13.6318
Beach nourishment          1.40612          11.5238         8.19549
Seawall                    7.2249            0.0620626      0.0085901
Building code              7.48629          19.0186         2.54045

--------------------  --------  --------
Total climate risk:   76.0742   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9189   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.17928           8.67047        7.35233
Beach nourishment          1.55347           6.72299        4.32772
Seawall                    7.98203           0.621468       0.0778584
Building code              8.27081          39.2619         4.74705

--------------------  --------  --------
Total climate risk:   157.048   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.771   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.17928          14.43          12.2363
Beach nourishment          1.55347          11.4252         7.35459
Seawall                    7.98203           0.0620626      0.00777529
Building code              8.27081          18.9582         2.29218

--------------------  --------  --------
Total climate risk:   75.8328   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9574   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148          13.8606        10.6498
Beach nourishment          1.71445          10.7904         6.29377
Seawall                    8.80916           0.175596       0.0199334
Building code              9.12786          29.4038         3.22132
```

(continues on next page)

(continued from previous page)

```
--------------------  --------  --------
Total climate risk:   117.615   (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3848  (USD bn)
--------------------  --------  --------
Net Present Values
```

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148           13.8606       10.6498
Beach nourishment          1.71445           10.7904        6.29377
Seawall                    8.80916            0.175596      0.0199334
Building code              9.12786           29.4038        3.22132


--------------------  --------  --------
Total climate risk:   117.615   (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3848  (USD bn)
--------------------  --------  --------
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992          10.5404        8.71168
Beach nourishment          1.59383           8.59532       5.39285
Seawall                    8.18941           0.0184996     0.00225897
Building code              8.48569           7.53759       0.88827Net Present Values


--------------------  --------  --------
Total climate risk:   30.1504   (USD bn)
Average annual risk:   3.37008  (USD bn)
Residual risk:         3.45852  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259             10.4109        8.26917
Beach nourishment          1.65849            8.18032       4.93239
Seawall                    8.52163            0.415328      0.0487381
Building code              8.82993           36.8546        4.17383

--------------------  --------  --------
Total climate risk:   147.418   (USD bn)
Average annual risk:   17.0232  (USD bn)
Residual risk:         91.5573  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288           15.0245       11.1883
Beach nourishment          1.76899           11.8507        6.69911
Seawall                    9.08939            0.108627      0.0119509
Building code              9.41823           24.4528        2.59633
```

(continues on next page)

(continued from previous page)

```
--------------------  -------  --------
Total climate risk:   97.8114  (USD bn)
Average annual risk:  11.2725  (USD bn)
Residual risk:        46.3747  (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148           14.0781        10.817
Beach nourishment          1.71445           10.968          6.39739
Seawall                    8.80916            0.175596       0.0199334
Building code              9.12786           29.5124         3.23322

--------------------  --------  --------
Total climate risk:   118.05    (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3155  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992           12.9104        10.6705
Beach nourishment          1.59383           10.0034         6.27629
Seawall                    8.18941            0.257712       0.0314689
Building code              8.48569           33.7365         3.97569

--------------------  --------  --------
Total climate risk:   134.946   (USD bn)
Average annual risk:   15.5605  (USD bn)
Residual risk:         78.038   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259              8.79133        6.98278
Beach nourishment          1.65849            6.82168        4.11319
Seawall                    8.52163            0.621468       0.0729283
Building code              8.82993           39.3223         4.45329


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.32205          5.31429         4.01972
Beach nourishment          1.74155          4.27155         2.45273
Seawall                    8.9484           0.00764219      0.000854029
Building code              9.27214          3.5502          0.382889--------------------
↪-  --------  --------
Total climate risk:   157.289   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.732   (USD bn)
--------------------  --------  --------
```

(continues on next page)

(continued from previous page)

```
Net Present Values--------------------  --------  --------
Total climate risk:   14.2008   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0571   (USD bn)
--------------------  --------  --------

Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148          14.1012        10.8347
Beach nourishment          1.71445          10.9632         6.39461
Seawall                    8.80916           0.0376243      0.00427104
Building code              9.12786          13.3845         1.46633

--------------------  --------  --------
Total climate risk:   53.5379   (USD bn)
Average annual risk:   6.15933  (USD bn)
Residual risk:        15.0513   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774            8.44955        6.08873
Beach nourishment          1.82807            6.4567         3.53197
Seawall                    9.39298            0.895618       0.0953497
Building code              9.7328            41.407          4.25438

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259             10.4834         8.32677
Beach nourishment          1.65849            8.23954        4.96809
Seawall                    8.52163            0.415328       0.0487381
Building code              8.82993           36.8908         4.17793


--------------------  --------  --------
Total climate risk:   165.628   (USD bn)
Average annual risk:   19.1818  (USD bn)
Residual risk:        108.419   (USD bn)
--------------------  --------  --------
--------------------  --------  --------
Total climate risk:   147.563   (USD bn)
Average annual risk:   17.0232  (USD bn)
Residual risk:         91.5342  (USD bn)
--------------------  --------  --------Net Present Values

Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.32205          5.5802         4.22086
Beach nourishment          1.74155          4.48867        2.5774
Seawall                    8.9484           0.00764219     0.000854029
Building code              9.27214          3.68297        0.397208
```

(continues on next page)

(continued from previous page)

```
--------------------  ---------  --------
Total climate risk:   14.7319    (USD bn)
Average annual risk:   1.57569   (USD bn)
Residual risk:         0.972395  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612           13.8606         8.90716
Beach nourishment          2.04988           10.7904         5.2639
Seawall                   10.5327             0.175596       0.0166716
Building code             10.9137            29.4038         2.69421

--------------------  --------  --------
Total climate risk:   117.615   (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3848  (USD bn)
```

```
--------------------  --------  --------
Net Present Values


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774            8.47373        6.10615
Beach nourishment          1.82807            6.47644        3.54276
Seawall                    9.39298            0.895618       0.0953497
Building code              9.7328            41.4191         4.25562Measure                 ␣
↪    Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259              8.71881        6.92518
Beach nourishment          1.65849            6.76247        4.07748
Seawall                    8.52163            0.621468       0.0729283
Building code              8.82993           39.286          4.44919




--------------------  --------  --------
Total climate risk:   165.676   (USD bn)
Average annual risk:   19.1818  (USD bn)
Residual risk:        108.411   (USD bn)
--------------------  --------  ----------------------------  --------  --------
Total climate risk:   157.144   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.755   (USD bn)
--------------------  --------  --------

Net Present ValuesNet Present Values


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.32205            7.60825        5.75487
Beach nourishment          1.74155            5.85798        3.36366
Seawall                    8.9484             1.82863        0.204353
```

(continues on next page)

(continued from previous page)

```
Building code              9.27214            44.9651         4.84948

--------------------  -------  --------
Total climate risk:   179.86   (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.6    (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612          14.3188         9.20163
Beach nourishment          2.04988          11.1409         5.4349
Seawall                   10.5327            0.0376243      0.00357216
Building code             10.9137           13.4931         1.23634

--------------------  --------  --------
Total climate risk:   53.9724   (USD bn)
Average annual risk:   6.15933  (USD bn)
Residual risk:        14.982    (USD bn)
--------------------  --------  --------

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774         1.39186         1.00297
Beach nourishment          1.82807         1.13491         0.62082
Seawall                    9.39298         0.000227424     2.42121e-05
Building code              9.7328          0.76062         0.0781501Net Present Values


--------------------  ---------  --------
Total climate risk:    3.04248   (USD bn)
Average annual risk:   0.260244  (USD bn)
Residual risk:        -0.245137  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288          14.575         10.8535
Beach nourishment          1.76899          11.5436         6.52553
Seawall                    9.08939           0.0620626      0.00682803
Building code              9.41823          19.0306         2.02062

--------------------  --------  --------
Total climate risk:   76.1225   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9112   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.36453          5.31429        3.89458
Beach nourishment          1.79751          4.27155        2.37637
Seawall                    9.23593          0.00764219     0.000827442
Building code              9.57007          3.5502         0.370969
```

(continues on next page)

(continued from previous page)

```
--------------------  --------  --------
Total climate risk:   14.2008   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0571   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774            8.44955        6.08873
Beach nourishment          1.82807            6.4567         3.53197
Seawall                    9.39298            0.895618       0.0953497
Building code              9.7328            41.407          4.25438

--------------------  --------  --------
Total climate risk:   165.628   (USD bn)
Average annual risk:   19.1818  (USD bn)
Residual risk:        108.419   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612             6.59956        4.24104
Beach nourishment          2.04988             5.16368        2.51902
Seawall                   10.5327              3.55475        0.337498
Building code             10.9137             48.016          4.3996

--------------------  --------  --------
Total climate risk:   192.064   (USD bn)
Average annual risk:   22.2359  (USD bn)
Residual risk:        128.73    (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288         14.3091        10.6555
Beach nourishment          1.76899         11.3265         6.40279
Seawall                    9.08939          0.0620626      0.00682803
Building code              9.41823         18.8979         2.00652


--------------------  --------  --------
Total climate risk:   75.5914   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9959   (USD bn)
--------------------  --------  --------
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.36453             7.87416        5.77059
Beach nourishment          1.79751             6.0751         3.37973
Seawall                    9.23593             1.82863        0.197991
Building code              9.57007            45.0978         4.71238
Net Present Values
```

(continues on next page)

(continued from previous page)

```
--------------------  -------  --------
Total climate risk:   180.391  (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.516  (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774        1.41604         1.02039
Beach nourishment          1.82807        1.15464         0.631618
Seawall                    9.39298        0.000227424     2.42121e-05
Building code              9.7328         0.77269         0.0793903

--------------------  ---------  --------
Total climate risk:    3.09076   (USD bn)
Average annual risk:   0.260244  (USD bn)
Residual risk:        -0.252837  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612            6.43034       4.1323
Beach nourishment          2.04988            5.02552       2.45161
Seawall                   10.5327             3.55475       0.337498
Building code             10.9137            47.9315        4.39186

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288            7.41063       5.51844
Beach nourishment          1.76899            5.87997       3.32392
Seawall                    9.08939            2.58858       0.284791
Building code              9.41823           46.6706        4.95535


--------------------  --------  --------
Total climate risk:   191.726   (USD bn)
Average annual risk:   22.2359  (USD bn)
Residual risk:        128.784   (USD bn)
--------------------  --------  ----------------------------  --------  --------
Total climate risk:   186.682   (USD bn)
Average annual risk:   21.5984  (USD bn)
Residual risk:        124.133   (USD bn)
--------------------  --------  --------
Net Present Values

Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.06742           8.67047       8.12282
Beach nourishment          1.40612           6.72299       4.78124
Seawall                    7.2249            0.621468      0.0860176
Building code              7.48629          39.2619        5.24451

--------------------  --------  --------
```

(continues on next page)

(continued from previous page)

```
Total climate risk:   157.048   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.771   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426         2.77518        1.92153
Beach nourishment          1.90253         2.24443        1.17971
Seawall                    9.77553         0.00264385     0.000270456
Building code             10.1292          1.68273        0.166127

--------------------  ---------  --------
Total climate risk:   6.73092    (USD bn)
Average annual risk:  0.678263   (USD bn)
Residual risk:        0.0259328  (USD bn)
--------------------  ---------  --------
Net Present Values


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612             7.59067         4.87796
Beach nourishment          2.04988             5.96389         2.90939
Seawall                   10.5327              1.31269         0.12463
Building code             10.9137             43.2513          3.96302Measure          ␣
↪      Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.13888        14.575         12.7977
Beach nourishment          1.50025        11.5436         7.69445
Seawall                    7.70855         0.0620626      0.00805114
Building code              7.98743        19.0306         2.38257



--------------------  --------  --------
Total climate risk:   173.005   (USD bn)
Average annual risk:   20.0179  (USD bn)
Residual risk:        114.887   (USD bn)
--------------------  --------  ----------------------------  --------  --------
Total climate risk:   76.1225   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9112   (USD bn)
--------------------  --------  --------

Net Present ValuesNet Present Values


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.06742          8.5496          8.00959
Beach nourishment          1.40612          6.6243          4.71105
Seawall                    7.2249           0.621468        0.0860176
Building code              7.48629         39.2016          5.23645

--------------------  --------  --------
```

(continues on next page)

(continued from previous page)

```
Total climate risk:   156.806   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.809   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426         2.70266        1.87132
Beach nourishment          1.90253         2.18521        1.14859
Seawall                    9.77553         0.00264385     0.000270456
Building code             10.1292          1.64652        0.162552


--------------------  ---------  --------
Total climate risk:   6.58607    (USD bn)
Average annual risk:  0.678263   (USD bn)
Residual risk:        0.0490347  (USD bn)
--------------------  ---------  --------Measure              Cost (USD bn)    ␣
↪Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148            6.59956        5.0708
Beach nourishment          1.71445            5.16368        3.01186
Seawall                    8.80916            3.55475        0.403528
Building code              9.12786           48.016          5.26038

Net Present Values
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.13888            7.14472        6.27348
Beach nourishment          1.50025            5.66285        3.77461
Seawall                    7.70855            2.58858        0.335806
Building code              7.98743           46.5378         5.82638



--------------------  --------  --------
Total climate risk:   192.064   (USD bn)
Average annual risk:   22.2359  (USD bn)
Residual risk:        128.73    (USD bn)
--------------------  --------  --------
--------------------  --------  --------
Total climate risk:   186.151   (USD bn)
Average annual risk:   21.5984  (USD bn)
Residual risk:        124.217   (USD bn)
--------------------  --------  --------Net Present Values
Net Present Values


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.06742        14.5509        13.6318
Beach nourishment          1.40612        11.5238         8.19549
Seawall                    7.2249          0.0620626      0.0085901
Building code              7.48629        19.0186         2.54045

--------------------  --------  --------
```

(continues on next page)

(continued from previous page)

```
Total climate risk:   76.0742   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9189   (USD bn)
--------------------  --------  --------
Net Present Values


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.13888             7.63243       6.70172
Beach nourishment          1.50025             5.87772       3.91783
Seawall                    7.70855             1.82863       0.237221
Building code              7.98743            44.9771        5.63099Measure          ␣
↪    Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148             7.42146       5.70231
Beach nourishment          1.71445             5.82573       3.39801
Seawall                    8.80916             1.31269       0.149014
Building code              9.12786            43.1668        4.72913



Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426          5.43516        3.76329
Beach nourishment          1.90253          4.37024        2.29707
Seawall                    9.77553          0.00764219     0.000781768
Building code             10.1292           3.61055        0.35645
--------------------  -------  --------
Total climate risk:   179.909  (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.593  (USD bn)
```

```
--------------------  -------  --------
--------------------  --------  --------
Total climate risk:   172.667   (USD bn)
Average annual risk:   20.0179  (USD bn)
Residual risk:        114.941   (USD bn)
--------------------  --------  --------

Net Present Values

--------------------  --------  --------
Total climate risk:   14.4422   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0186   (USD bn)
--------------------  --------  --------Net Present Values

Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.17928            8.67047       7.35233
Beach nourishment          1.55347            6.72299       4.32772
Seawall                    7.98203            0.621468      0.0778584
Building code              8.27081           39.2619        4.74705
```

(continues on next page)

(continued from previous page)

```
--------------------  --------  --------
Total climate risk:   157.048   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.771   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.13888            7.7533          6.80785
Beach nourishment          1.50025            5.97641         3.98361
Seawall                    7.70855            1.82863         0.237221
Building code              7.98743           45.0375          5.63855

--------------------  -------  --------
Total climate risk:   180.15   (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.554  (USD bn)
--------------------  -------  --------

Net Present ValuesMeasure              Cost (USD bn)    Benefit (USD bn)    Benefit/
↪Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992         10.5404         8.71168
Beach nourishment          1.59383          8.59532        5.39285
Seawall                    8.18941          0.0184996      0.00225897
Building code              8.48569          7.53759        0.88827



--------------------  --------  --------
Total climate risk:   30.1504   (USD bn)
Average annual risk:   3.37008  (USD bn)
Residual risk:         3.45852  (USD bn)
--------------------  --------  --------
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426         2.77518        1.92153
Beach nourishment          1.90253         2.24443        1.17971
Seawall                    9.77553         0.00264385     0.000270456
Building code             10.1292          1.68273        0.166127Net Present Values


--------------------  ---------  --------
Total climate risk:   6.73092    (USD bn)
Average annual risk:  0.678263   (USD bn)
Residual risk:        0.0259328  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.17928        14.43          12.2363
Beach nourishment          1.55347        11.4252         7.35459
Seawall                    7.98203         0.0620626      0.00777529
Building code              8.27081        18.9582         2.29218
```

(continues on next page)

(continued from previous page)

```
--------------------  --------  --------
Total climate risk:   75.8328   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9574   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.13888           14.9037        13.0863
Beach nourishment          1.50025           11.752          7.83335
Seawall                    7.70855            0.108627       0.0140917
Building code              7.98743           24.3925         3.05386


--------------------  -------  --------
Total climate risk:   97.5699  (USD bn)
Average annual risk:  11.2725  (USD bn)
Residual risk:        46.4132  (USD bn)
--------------------  -------  --------Measure              Cost (USD bn)    Benefit␣
↪(USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992          10.5646         8.73166
Beach nourishment          1.59383           8.61505        5.40524
Seawall                    8.18941           0.0184996      0.00225897
Building code              8.48569           7.54966        0.889693

Net Present Values

--------------------  --------  --------
Total climate risk:   30.1986   (USD bn)
Average annual risk:   3.37008  (USD bn)
Residual risk:         3.45082  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426          5.36264         3.71308
Beach nourishment          1.90253          4.31103         2.26595
Seawall                    9.77553          0.00764219      0.000781768
Building code             10.1292           3.57434         0.352875

--------------------  --------  --------
Total climate risk:   14.2973   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0417   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288            7.63243        5.68361
Beach nourishment          1.76899            5.87772        3.32264
Seawall                    9.08939            1.82863        0.201183
Building code              9.41823           44.9771         4.77554
```

(continues on next page)

(continued from previous page)

```
--------------------  -------  --------
Total climate risk:   179.909  (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.593  (USD bn)
--------------------  -------  --------Measure              Cost (USD bn)    Benefit␣
↪(USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.20992           12.8863        10.6505
Beach nourishment         1.59383            9.98362        6.2639
Seawall                   8.18941            0.257712       0.0314689
Building code             8.48569           33.7244         3.97427

Net Present Values

--------------------  --------  --------
Total climate risk:   134.898   (USD bn)
Average annual risk:   15.5605  (USD bn)
Residual risk:         78.0457  (USD bn)
--------------------  --------  --------

Net Present ValuesMeasure              Cost (USD bn)    Benefit (USD bn)    Benefit/
↪Cost
-----------------  ---------------  ------------------  --------------
Mangroves                 1.259             10.4834         8.32677
Beach nourishment         1.65849            8.23954        4.96809
Seawall                   8.52163            0.415328       0.0487381
Building code             8.82993           36.8908         4.17793


--------------------  --------  --------
Total climate risk:   147.563   (USD bn)
Average annual risk:   17.0232  (USD bn)
Residual risk:         91.5342  (USD bn)
--------------------  --------  --------
Net Present Values
```

The output of `CostBenefit.calc` is rather complex in its structure. The metrics dictionary inherits this complexity.

```
#Top level metrics keys
macro_metrics = output_cb.uncertainty_metrics
macro_metrics
```

```
['imp_meas_present',
 'imp_meas_future',
 'tot_climate_risk',
 'benefit',
 'cost_ben_ratio']
```

```
# The benefits and cost_ben_ratio are available for each measure
output_cb.get_uncertainty(metric_list=['benefit', 'cost_ben_ratio']).tail()
```

```
    Mangroves Benef  Beach nourishment Benef  Seawall Benef  \
45     8.670468e+09             6.722992e+09   6.214684e+08
46     8.549601e+09             6.624301e+09   6.214684e+08
```

(continues on next page)

(continued from previous page)

```
47     1.455086e+10             1.152385e+10   6.206260e+07
48     8.670468e+09             6.722992e+09   6.214684e+08
49     1.443000e+10             1.142516e+10   6.206260e+07

    Building code Benef  Mangroves CostBen  Beach nourishment CostBen  \
45         3.926190e+10           0.123110                   0.209151
46         3.920155e+10           0.124850                   0.212267
47         1.901856e+10           0.073358                   0.122018
48         3.926190e+10           0.136011                   0.231069
49         1.895821e+10           0.081724                   0.135970

    Seawall CostBen  Building code CostBen
45        11.625533               0.190676
46        11.625533               0.190969
47       116.413127               0.393631
48        12.843826               0.210657
49       128.612593               0.436265
```

```
# The impact_meas_present and impact_meas_future provide values of the cost_meas,␣
↪risk_transf, risk,
# and cost_ins for each measure
output_cb.get_uncertainty(metric_list=['imp_meas_present']).tail()
```

```
    no measure - risk - present  no measure - risk_transf - present  \
45                1.040893e+08                                 0.0
46                8.010560e+07                                 0.0
47                1.040893e+08                                 0.0
48                1.040893e+08                                 0.0
49                8.010560e+07                                 0.0

    no measure - cost_meas - present  no measure - cost_ins - present  \
45                                 0                                0
46                                 0                                0
47                                 0                                0
48                                 0                                0
49                                 0                                0

    Mangroves - risk - present  Mangroves - risk_transf - present  \
45                5.197409e+07                                  0
46                3.999849e+07                                  0
47                5.197409e+07                                  0
48                5.197409e+07                                  0
49                3.999849e+07                                  0

    Mangroves - cost_meas - present  Mangroves - cost_ins - present  \
45                     1.311768e+09                               1
46                     1.311768e+09                               1
47                     1.311768e+09                               1
48                     1.311768e+09                               1
49                     1.311768e+09                               1

    Beach nourishment - risk - present  \
45                        6.153578e+07
46                        4.735703e+07
47                        6.153578e+07
48                        6.153578e+07
```

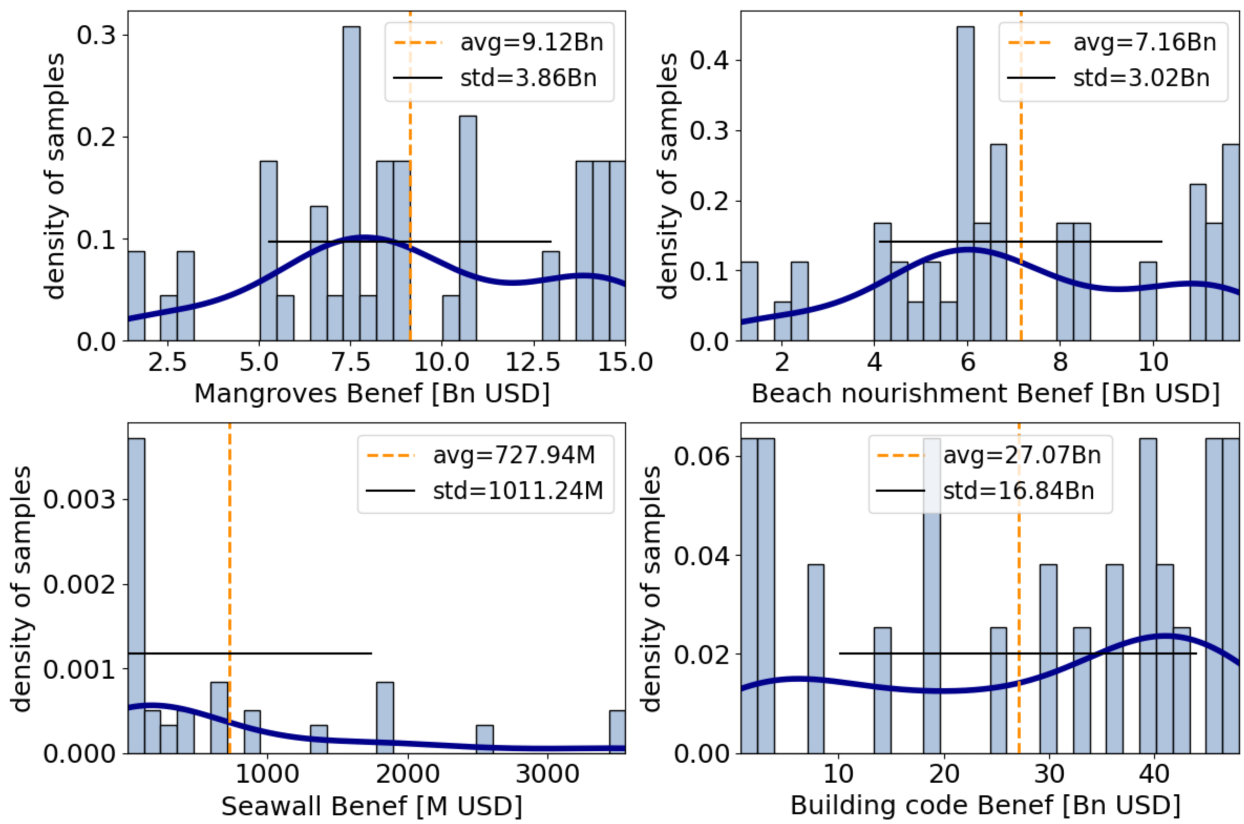
(continues on next page)

(continued from previous page)

```
49                         4.735703e+07

    Beach nourishment - risk_transf - present  \
45                                        0
46                                        0
47                                        0
48                                        0
49                                        0

    Beach nourishment - cost_meas - present  \
45                              1.728000e+09
46                              1.728000e+09
47                              1.728000e+09
48                              1.728000e+09
49                              1.728000e+09

    Beach nourishment - cost_ins - present  Seawall - risk - present  \
45                                       1              1.040893e+08
46                                       1              8.010560e+07
47                                       1              1.040893e+08
48                                       1              1.040893e+08
49                                       1              8.010560e+07

    Seawall - risk_transf - present  Seawall - cost_meas - present  \
45                                0                   8.878779e+09
46                                0                   8.878779e+09
47                                0                   8.878779e+09
48                                0                   8.878779e+09
49                                0                   8.878779e+09

    Seawall - cost_ins - present  Building code - risk - present  \
45                             1                    7.806698e+07
46                             1                    6.007920e+07
47                             1                    7.806698e+07
48                             1                    7.806698e+07
49                             1                    6.007920e+07

    Building code - risk_transf - present  \
45                                      0
46                                      0
47                                      0
48                                      0
49                                      0

    Building code - cost_meas - present  Building code - cost_ins - present
45                         9.200000e+09                                  1
46                         9.200000e+09                                  1
47                         9.200000e+09                                  1
48                         9.200000e+09                                  1
49                         9.200000e+09                                  1
```

We can plot the distributions for the top metrics or our choice.

```
# tot_climate_risk and benefit
output_cb.plot_uncertainty(metric_list=['benefit'], figsize=(12,8));
```



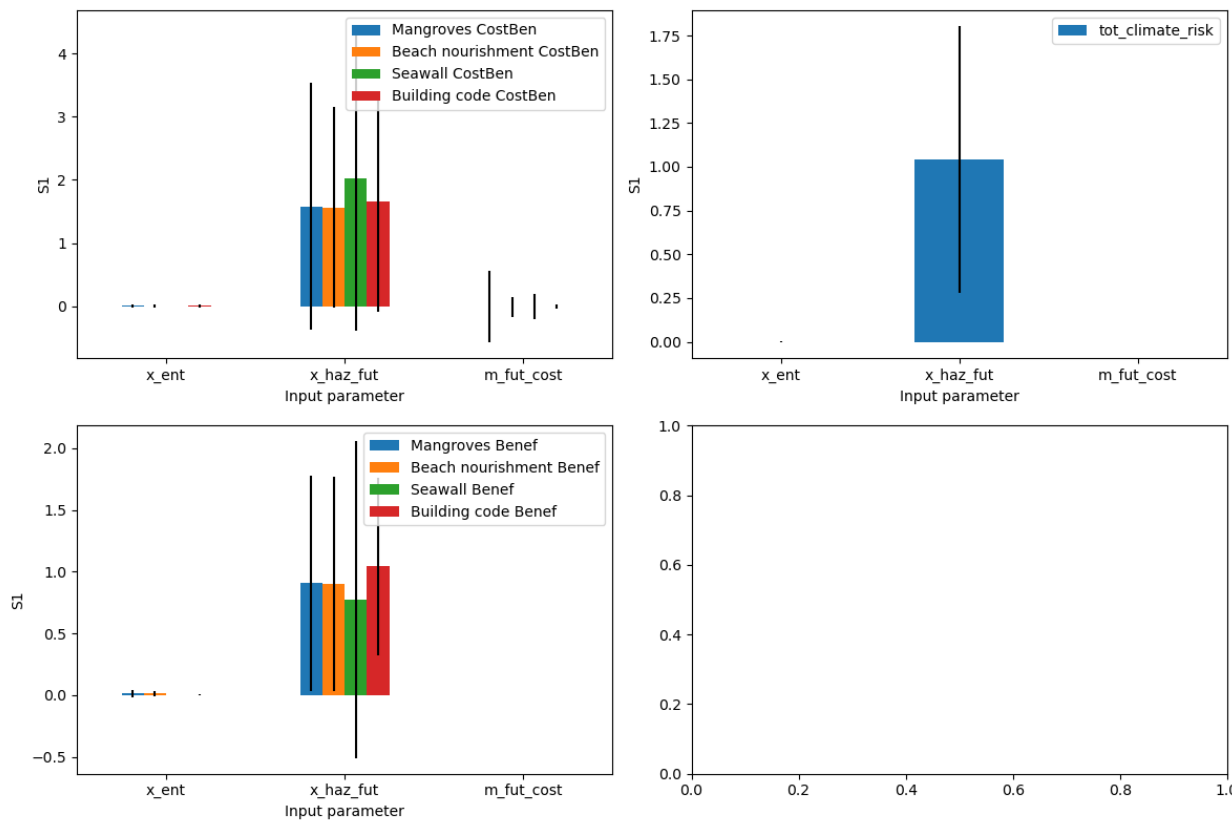
Analogously to the impact example, now that we have a metric distribution, we can compute the sensitivity indices. Since we used the default sampling method, we can use the default sensitivity analysis method. However, since we used `calc_second_order = False` for the sampling, we need to specify the same for the sensitivity analysis.

```
output_cb = unc_cb.sensitivity(output_cb, sensitivity_kwargs={'calc_second_order
↪':False})
```

The sensitivity indices can be plotted. For the default method 'sobol', by default the 'S1' sensitivity index is plotted.

Note that since we have quite a few measures, the plot must be adjusted a bit or dropped. Also see that for many metrics, the sensitivity to certain uncertainty parameters appears to be 0. However, this result is to be treated with care. Indeed, we used for demonstration purposes a rather too low number of samples, which is indicated by large confidence intervals (vertical black lines) for most sensitivity indices. For a more robust result the analysis should be repeated with more samples.

```
#plot only certain metrics
axes = output_cb.plot_sensitivity(metric_list=['cost_ben_ratio','tot_climate_risk',
↪'benefit'], figsize=(12,8));
```



11.2.8 Advanced examples

Coupled variables

In this example, we show how you can define correlated input variables. Suppose your exposures and hazards are conditioned on the same Shared Socio-economic Pathway (SSP). Then, you want that only exposures and hazard belonging to the same SSP are present in each sample.

In order to achieve this, you must simply define an uncertainty parameter that shares the same name and the same distribution for both the exposures and the hazard uncertainty variables.

Many scenarios of hazards and exposures

In this example we look at the case where many scenarios are tested in the uncertainty analysis. For instance, suppose you have data for different Shared Socio-economic Pathways (SSP) and different Climate Change projections. From the SSPs, you have a number of Exposures, saved to files. From the climate projections, you have a number of Hazards, saved to file.

The task is to sample from the SSPs and the Climate change scenarios for the uncertainty and sensitivity analysis efficiently.

For demonstration purposes, we will use below as exposures files the litpop for three countries, and for tha hazard files the winter storms for the same three countries. Instead of having SSPs, we now want to only combine exposures and hazards of the same countries.

```
from climada.util.api_client import Client
client = Client()
```

```
def get_litpop(iso):
    return client.get_litpop(country=iso)


def get_ws(iso):
    properties = {
        'country_iso3alpha': iso,
    }
    return client.get_hazard('storm_europe', properties=properties)
```

```
#Define list of exposures and/or of hazard files

exp_list = [get_litpop(iso) for iso in ['CHE', 'DEU', 'ITA']]
haz_list = [get_ws(iso) for iso in ['CHE', 'DEU', 'ITA']]
for exp, haz in zip(exp_list, haz_list):
    exp.gdf['impf_WS'] = 1
    exp.assign_centroids(haz)
```

```
#Define the input variable
from climada.entity import ImpactFuncSet, Exposures
from climada.entity.impact_funcs.storm_europe import ImpfStormEurope
from climada.hazard import Hazard
from climada.engine.unsequa import InputVar
import scipy as sp
import copy

def exp_func(cnt, x_exp, exp_list=exp_list):
    exp = exp_list[int(cnt)].copy()
    exp.gdf.value *= x_exp
    return exp

exp_distr = {"x_exp": sp.stats.uniform(0.9, 0.2),
             "cnt": sp.stats.randint(low=0, high=len(exp_list))  #use the same␣
↪parameter name accross input variables
              }
exp_iv = InputVar(exp_func, exp_distr)


def haz_func(cnt, i_haz, haz_list=haz_list):
    haz = copy.deepcopy(haz_list[int(cnt)])  #use the same parameter name accross␣
↪input variables
```

(continues on next page)

(continued from previous page)

```
    haz.intensity *= i_haz
    return haz

haz_distr = {"i_haz": sp.stats.norm(1, 0.2),
             "cnt": sp.stats.randint(low=0, high=len(haz_list))
              }
haz_iv = InputVar(haz_func, haz_distr)

impf = ImpfStormEurope.from_schwierz()
impf_set = ImpactFuncSet()
impf_set.append(impf)
impf_iv = InputVar.impfset([impf_set], bounds_mdd = [0.9, 1.1])
```

```
from climada.engine.unsequa import CalcImpact

calc_imp = CalcImpact(exp_iv, impf_iv, haz_iv)
```

```
2024-01-25 15:38:30,713 - climada.engine.unsequa.calc_base - WARNING -

The input parameter cnt is shared among at least 2 input variables. Their uncertainty␣
↪is thus computed with the same samples for this input paramter.
```

```
output_imp = calc_imp.make_sample(N=2**2, sampling_kwargs={'skip_values': 2**3})
```

```
# as we can see, there is only a single input parameter "cnt" to select the country␣
↪for both the exposures and the hazard
output_imp.samples_df.tail()
```

	x_exp	cnt	MDD	i_haz
35	0.9875	0.0	1.0375	1.097755
36	1.0625	1.0	1.0375	1.097755
37	1.0625	0.0	0.9375	1.097755
38	1.0625	0.0	1.0375	1.097755
39	1.0625	0.0	1.0375	1.097755

```
output_imp = calc_imp.uncertainty(output_imp)
```

```
output_imp.aai_agg_unc_df.tail()
```

Input variable: Repeated loading of files made efficient

Loading Hazards or Exposures from file is a rather lengthy operation. Thus, we want to minimize the reading operations, ideally reading each file only once. Simultaneously, Hazard and Exposures can be large in memory, and thus we would like to have at most one of each loaded at a time. Thus, we do not want to use the list capacity from the helper method InputVar.exposures and InputVar.hazard.

For demonstration purposes, we will use below as exposures files the litpop for three countries, and for tha hazard files the winter storms for the same three countries. Note that this does not make a lot of sense for an uncertainty analysis. For your use case, please replace the set of exposures and/or hazard files with meaningful sets, for instance sets of exposures for different resolutions or hazards for different model runs.

```
from climada.util.api_client import Client
client = Client()
```

```
def get_litpop_path(iso):
    properties = {
        'country_iso3alpha': iso,
        'res_arcsec': '150',
        'exponents': '(1,1)',
        'fin_mode': 'pc'
    }
    litpop_datasets = client.list_dataset_infos(data_type='litpop',␣
↪properties=properties)
    ds = litpop_datasets[0]
    download_dir, ds_files = client.download_dataset(ds)
    return ds_files[0]

def get_ws_path(iso):
    properties = {
        'country_iso3alpha': iso,
    }
    hazard_datasets = client.list_dataset_infos(data_type='storm_europe',␣
↪properties=properties)
    ds = hazard_datasets[0]
    download_dir, ds_files = client.download_dataset(ds)
    return ds_files[0]
```

```
#Define list of exposures and/or of hazard files

f_exp_list = [get_litpop_path(iso) for iso in ['CHE', 'DEU', 'ITA']]
f_haz_list = [get_ws_path(iso) for iso in ['CHE', 'DEU', 'ITA']]
```

```
#Define the input variable for the loading files
#The trick is to not reload a file if it is already in memory. This is done using a␣
↪global variable.
from climada.entity import ImpactFunc, ImpactFuncSet, Exposures
from climada.hazard import Hazard
from climada.engine.unsequa import InputVar
import scipy as sp
import copy

def exp_func(f_exp, x_exp, filename_list=f_exp_list):
    filename = filename_list[int(f_exp)]
    global exp_base
    if 'exp_base' in globals():
        if isinstance(exp_base, Exposures):
            if exp_base.gdf.filename != str(filename):
                exp_base = Exposures.from_hdf5(filename)
                exp_base.gdf.filename = str(filename)
    else:
        exp_base = Exposures.from_hdf5(filename)
        exp_base.gdf.filename = str(filename)

    exp = exp_base.copy()
    exp.gdf.value *= x_exp
    return exp
```

(continues on next page)

(continued from previous page)

```
exp_distr = {"x_exp": sp.stats.uniform(0.9, 0.2),
             "f_exp": sp.stats.randint(low=0, high=len(f_exp_list))
              }
exp_iv = InputVar(exp_func, exp_distr)


def haz_func(f_haz, i_haz, filename_list=f_haz_list):
    filename = filename_list[int(f_haz)]
    global haz_base
    if 'haz_base' in globals():
        if isinstance(haz_base, Hazard):
            if haz_base.filename != str(filename):
                haz_base = Hazard.from_hdf5(filename)
                haz_base.filename = str(filename)
    else:
        haz_base = Hazard.from_hdf5(filename)
        haz_base.filename = str(filename)

    haz = copy.deepcopy(haz_base)
    haz.intensity *= i_haz
    return haz

haz_distr = {"i_haz": sp.stats.norm(1, 0.2),
             "f_haz": sp.stats.randint(low=0, high=len(f_haz_list))
              }
haz_iv = InputVar(haz_func, haz_distr)


def impf_func(G=1, v_half=84.7, vmin=25.7, k=3, _id=1):

    def xhi(v, v_half, vmin):
        return max([(v - vmin), 0]) / (v_half - vmin)

    def sigmoid_func(v, G, v_half, vmin, k):
        return G * xhi(v, v_half, vmin)**k / (1 + xhi(v, v_half, vmin)**k)

    #In-function imports needed only for parallel computing on Windows
    import numpy as np
    from climada.entity import ImpactFunc, ImpactFuncSet
    imp_fun = ImpactFunc()
    imp_fun.haz_type = 'WS'
    imp_fun.id = _id
    imp_fun.intensity_unit = 'm/s'
    imp_fun.intensity = np.linspace(0, 150, num=100)
    imp_fun.mdd = np.repeat(1, len(imp_fun.intensity))
    imp_fun.paa = np.array([sigmoid_func(v, G, v_half, vmin, k) for v in imp_fun.
↪intensity])
    imp_fun.check()
    impf_set = ImpactFuncSet()
    impf_set.append(imp_fun)
    return impf_set

impf_distr = {
    "G": sp.stats.truncnorm(0.5, 1.5),
    "v_half": sp.stats.uniform(35, 65),
    "vmin": sp.stats.uniform(0, 15),
    "k": sp.stats.uniform(1, 4)
```

(continues on next page)

(continued from previous page)

```
    }
impf_iv = InputVar(impf_func, impf_distr)
```

```
from climada.engine.unsequa import CalcImpact

calc_imp = CalcImpact(exp_iv, impf_iv, haz_iv)
```

Now that the samples have been generated, it is crucial to oder the samples in order to minimize the number of times files have to be loaded. In this case, loading the hazards take more time than loading the exposures. We thus sort first by hazards (which then each have to be loaded one single time), and then by exposures (which have to be each loaded once for each hazard).

```
# Ordering of the samples by hazard first and exposures second
output_imp = calc_imp.make_sample(N=2**2, sampling_kwargs={'skip_values': 2**3})
output_imp.order_samples(by=['f_haz', 'f_exp'])
```

We can verify how the samples are ordered. In the graph below, it is confirmed that the hazard are ordered, and thus the hazards will be loaded once each. The exposures on the other changes at most once per hazard.

```
import matplotlib.pyplot as plt
e = output_imp.samples_df['f_exp'].values
h = output_imp.samples_df['f_haz'].values
```

Note that due to the very small number of samples chosen here for illustrative purposes, not all combinations of hazard and exposures are part of the samples. This is due to the nature of the Sobol sequence (default sampling method).

```
plt.plot(e, label='exposures');
plt.plot(h, label='hazards');
plt.xlabel('samples');
plt.ylabel('file number');
plt.title('Order of exposures and hazards files in samples');
plt.legend(loc='upper right');
```

```
output_imp = calc_imp.uncertainty(output_imp)
```

11.3 Helper methods for InputVar

This tutorial complements the general tutorial on the uncertainty and sensitivity analysis module *unsequa*.

The InputVar class provides a few helper methods to generate generic uncertainty input variables for exposures, impact function sets, hazards, and entities (including measures cost and disc rates).

- *Exposures*
- *Hazard*
- *ImpactFuncSet*
- *Entity*
- *Entity Future*

```
import warnings
warnings.filterwarnings('ignore') #Ignore warnings for making the tutorial's pdf.
```

11.3.1 Exposures

The following types of uncertainties can be added:

- ET: scale the total value (homogeneously)

  The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_totvalue

- EN: mutliplicative noise (inhomogeneous)

  The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds_noise. EN is the value of the seed for the uniform random number generator.

- EL: sample uniformly from exposure list

  From the provided list of exposure is elements are uniformly sampled. For example, LitPop instances with different exponents.

If a bounds is None, this parameter is assumed to have no uncertainty.

Example: single exposures

```
#Define the base exposure
from climada.util.constants import EXP_DEMO_H5
from climada.entity import Exposures
exp_base = Exposures.from_hdf5(EXP_DEMO_H5)
```

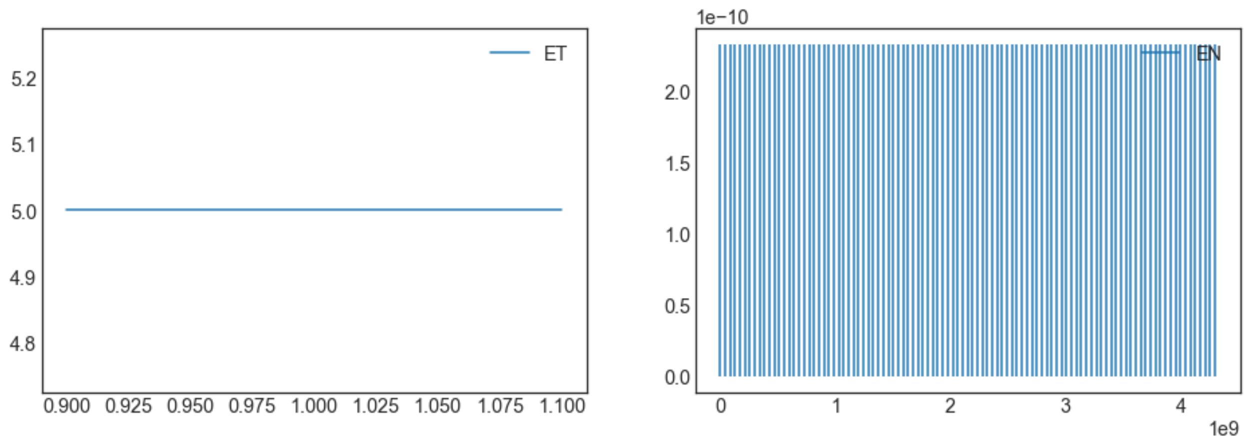
```
2022-07-07 15:13:32,000 - climada.entity.exposures.base - INFO - Reading /Users/
↪ckropf/climada/demo/data/exp_demo_today.h5
```

```
from climada.engine.unsequa import InputVar
bounds_totval = [0.9, 1.1] #+- 10% noise on the total exposures value
bounds_noise = [0.9, 1.2] #-10% - +20% noise each exposures point
exp_iv = InputVar.exp([exp_base], bounds_totval, bounds_noise)
```

```
#The difference in total value between the base exposure and the average input␣
↪uncertainty exposure
#due to the random noise on each exposures point (the average change in the total␣
↪value is 1.0).
avg_exp = exp_iv.evaluate()
(sum(avg_exp.gdf['value']) - sum(exp_base.gdf['value'])) / sum(exp_base.gdf['value'])
```

```
0.03700231587024304
```

```
#The values for EN are seeds for the random number generator for the noise sampling␣
↪and
#thus are uniformly sampled numbers between (0, 2**32-1)
exp_iv.plot();
```



Example: list of litpop exposures with different exponents

```
#Define a generic method to make litpop instances with different exponent pairs.
from climada.entity import LitPop
def generate_litpop_base(impf_id, value_unit, haz, assign_centr_kwargs,
                          choice_mn, **litpop_kwargs):
    #In-function imports needed only for parallel computing on Windows
    from climada.entity import LitPop
    litpop_base = []
    for [m, n] in choice_mn:
        print('\n Computing litpop for m=%d, n=%d \n' %(m, n))
        litpop_kwargs['exponents'] = (m, n)
        exp = LitPop.from_countries(**litpop_kwargs)
        exp.gdf['impf_' + haz.haz_type] = impf_id
        exp.gdf.drop('impf_', axis=1, inplace=True)
        if value_unit is not None:
            exp.value_unit = value_unit
        exp.assign_centroids(haz, **assign_centr_kwargs)
        litpop_base.append(exp)
    return litpop_base
```

```
#Define the parameters of the LitPop instances
tot_pop = 11.317e6
impf_id = 1
value_unit = 'people'
litpop_kwargs = {
    'countries' : ['CUB'],
    'res_arcsec' : 150,
    'reference_year' : 2020,
    'fin_mode' : 'norm',
    'total_values' : [tot_pop]
}
assign_centr_kwargs={}

# The hazard is needed to assign centroids
from climada.util.constants import HAZ_DEMO_H5
from climada.hazard import Hazard
haz = Hazard.from_hdf5(HAZ_DEMO_H5)
```

```
2022-07-07 15:13:32,787 - climada.hazard.base - INFO - Reading /Users/ckropf/climada/
↪demo/data/tc_fl_1990_2004.h5
```

```
#Generate the LitPop list

choice_mn = [[0, 0.5], [0, 1], [0, 2]] #Choice of exponents m,n

litpop_list = generate_litpop_base(impf_id, value_unit, haz, assign_centr_kwargs,␣
↪choice_mn, **litpop_kwargs)
```

```
 Computing litpop for m=0, n=0

2022-07-07 15:13:33,055 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:13:34,051 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,082 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,109 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,135 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,163 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,188 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,223 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,289 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,316 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,325 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:13:34,326 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,355 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,385 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,410 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,435 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,459 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,487 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,508 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,519 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:13:34,520 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,543 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:13:34,570 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,597 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,621 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,645 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,685 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,710 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,742 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,768 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,791 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,830 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,862 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,899 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,933 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,955 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,981 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,019 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,043 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,068 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,090 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,119 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,142 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,173 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:13:35,173 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:13:35,174 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:13:35,174 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:13:35,175 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:13:35,179 - climada.entity.exposures.base - INFO - Matching 5524␣
↪exposures with 2500 centroids.
2022-07-07 15:13:35,181 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:13:35,189 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 332 coordinates.

 Computing litpop for m=0, n=1
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:13:35,416 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:13:36,256 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,284 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,309 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,335 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,367 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,392 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,424 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,494 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,519 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,529 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:13:36,530 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,556 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,586 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,610 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,637 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,666 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,691 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,716 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,725 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:13:36,726 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,754 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,786 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,818 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,843 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,872 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,909 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,936 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:13:36,965 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

```
2022-07-07 15:13:36,989 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,013 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,052 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,087 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,123 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,156 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,177 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,201 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,241 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,263 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,288 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,311 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,343 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,367 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,400 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:13:37,400 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:13:37,401 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:13:37,401 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:13:37,402 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:13:37,406 - climada.entity.exposures.base - INFO - Matching 5524␣
↪exposures with 2500 centroids.
2022-07-07 15:13:37,407 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:13:37,415 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 332 coordinates.

 Computing litpop for m=0, n=2

2022-07-07 15:13:37,637 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:13:38,561 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,589 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,615 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,638 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:13:38,665 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,689 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,720 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,784 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,808 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,820 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:13:38,820 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,844 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,873 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,897 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,920 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,944 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,968 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,990 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,999 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:13:39,000 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,022 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,047 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,074 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,097 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,123 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,161 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,187 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,217 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,242 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,265 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,304 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,334 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:13:39,370 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,402 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,423 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,448 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,487 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,510 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,532 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,554 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,580 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,603 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,633 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:13:39,634 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:13:39,634 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:13:39,635 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:13:39,635 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:13:39,639 - climada.entity.exposures.base - INFO - Matching 5524␣
↪exposures with 2500 centroids.
2022-07-07 15:13:39,641 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:13:39,649 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 332 coordinates.
```

```python
from climada.engine.unsequa import InputVar
bounds_totval = [0.9, 1.1] #+- 10% noise on the total exposures value
litpop_iv = InputVar.exp(exp_list = litpop_list,
                         bounds_totval=bounds_totval)
```

```python
# To choose n=0.5, we have to set EL=1 (the index of 0.5 in choice_n = [0, 0.5, 1, 2])
pop_half = litpop_iv.evaluate(ET=1, EL=1)
```

```python
pop_half.gdf.tail()
```

```
           value                    geometry   latitude  longitude  region_id  \
5519   92.974926  POINT (-80.52083 23.18750)  23.187500 -80.520833        192
5520  131.480741  POINT (-80.47917 23.18750)  23.187500 -80.479167        192
5521   77.695093  POINT (-80.68750 23.18750)  23.187500 -80.687500        192
5522   43.122163  POINT (-80.89583 23.14583)  23.145833 -80.895833        192
5523  106.033524  POINT (-80.85417 23.14583)  23.145833 -80.854167        192

      impf_TC  centr_TC
5519        1       619
5520        1       619
5521        1       618
5522        1       617
```

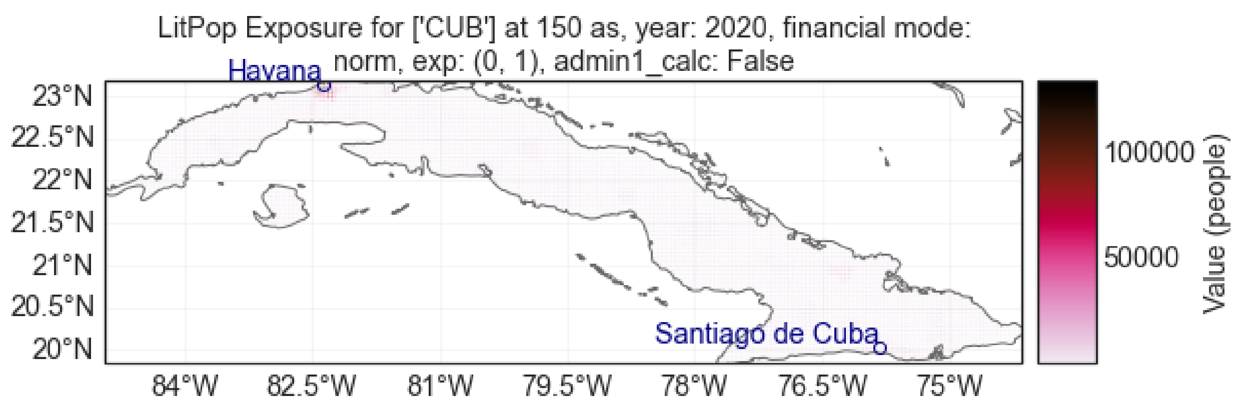
(continues on next page)

(continued from previous page)

```
5523        1       617
```

```
pop_half.plot_hexbin();
```

```
2022-07-07 15:13:39,690 - climada.util.plot - WARNING - Error parsing coordinate␣
↪system 'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
↪AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0],UNIT[
↪"degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS[
↪"Longitude",EAST],AUTHORITY["EPSG","4326"]]'. Using projection PlateCarree in plot.
```



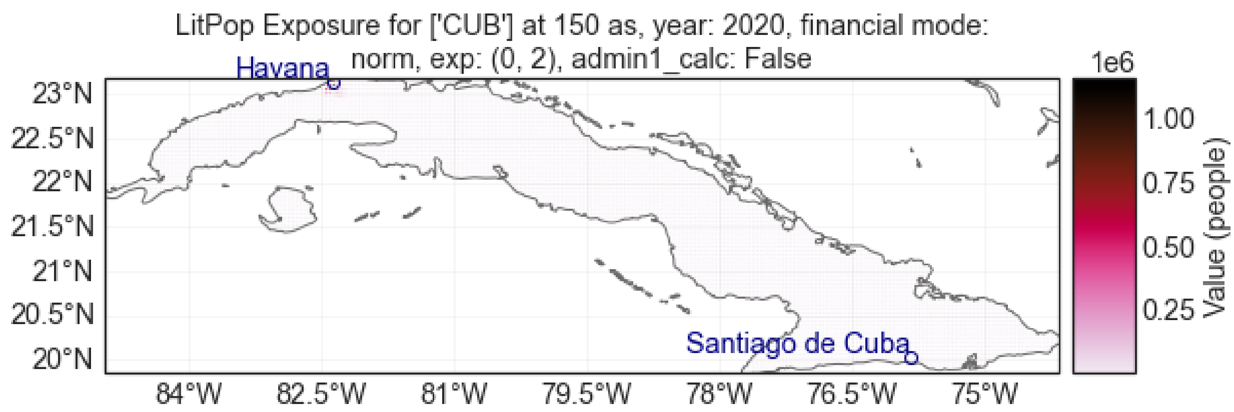

```
# To choose n=1, we have to set EL=2 (the index of 1 in choice_n = [0, 0.5, 1, 2])
pop_one = litpop_iv.evaluate(ET=1, EL=2)
```

```
pop_one.gdf.tail()
```

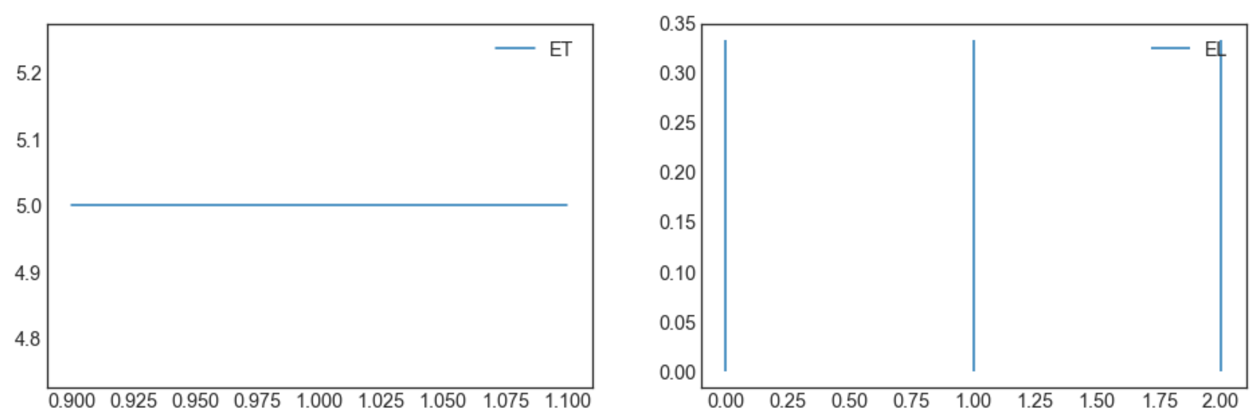
```
         value                    geometry   latitude  longitude  region_id  \
5519  0.567593  POINT (-80.52083 23.18750)  23.187500 -80.520833        192
5520  1.135089  POINT (-80.47917 23.18750)  23.187500 -80.479167        192
5521  0.396363  POINT (-80.68750 23.18750)  23.187500 -80.687500        192
5522  0.122097  POINT (-80.89583 23.14583)  23.145833 -80.895833        192
5523  0.738231  POINT (-80.85417 23.14583)  23.145833 -80.854167        192

      impf_TC  centr_TC
5519        1       619
5520        1       619
5521        1       618
5522        1       617
5523        1       617
```

```
pop_one.plot_hexbin();
```

```
2022-07-07 15:13:45,584 - climada.util.plot - WARNING - Error parsing coordinate␣
↪system 'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
↪AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0],UNIT[
↪"degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS[
↪"Longitude",EAST],AUTHORITY["EPSG","4326"]]'. Using projection PlateCarree in plot.
```




```
#The values for EN are seeds for the random number generator for the noise sampling␣
↪and
#thus are uniformly sampled numbers between (0, 2**32-1)
litpop_iv.plot();
```



11.3.2 Hazard

The following types of uncertainties can be added:

- HE: sub-sampling events from the total event set

  For each sub-sample, n_ev events are sampled with replacement. HE is the value of the seed for the uniform random number generator.

- HI: scale the intensity of all events (homogeneously)

  The instensity of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_int

- HA: scale the fraction of all events (homogeneously)

  The fraction of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_frac

- HF: scale the frequency of all events (homogeneously)

  The frequency of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_freq

- HL: sample uniformly from hazard list

  From the provided list of hazard is elements are uniformly sampled. For example, Hazards outputs from dynamical models for different input factors.

If a bounds is None, this parameter is assumed to have no uncertainty.

```
#Define the base exposure
from climada.util.constants import HAZ_DEMO_H5
from climada.hazard import Hazard
haz_base = Hazard.from_hdf5(HAZ_DEMO_H5)
```

```
2022-07-07 15:13:51,145 - climada.hazard.base - INFO - Reading /Users/ckropf/climada/
↪demo/data/tc_fl_1990_2004.h5
```

```
from climada.engine.unsequa import InputVar
bounds_freq = [0.9, 1.1] #+- 10% noise on the frequency of all events
bounds_int = None #No uncertainty on the intensity
n_ev = None
haz_iv = InputVar.haz([haz_base], n_ev=n_ev, bounds_freq=bounds_freq, bounds_
↪int=bounds_int)
```

```
#The difference in frequency for HF=1.1 is indeed 10%.
haz_high_freq = haz_iv.evaluate(HE=n_ev, HI=None, HF = 1.1)
(sum(haz_high_freq.frequency) - sum(haz_base.frequency)) / sum(haz_base.frequency)
```

```
0.10000000000000736
```

```
bounds_freq = [0.9, 1.1] #+- 10% noise on the frequency of all events
bounds_int = None #No uncertainty on the intensity
bounds_frac = [0.7, 1.1] #noise on the fraction of all events
n_ev = round(0.8 * haz_base.size) #sub-sample with re-draw events to obtain hazards␣
↪with n=0.8*tot_number_events
haz_iv = InputVar.haz(
    [haz_base], n_ev=n_ev, bounds_freq=bounds_freq, bounds_int=bounds_int, bounds_
↪frac=bounds_frac
)
```

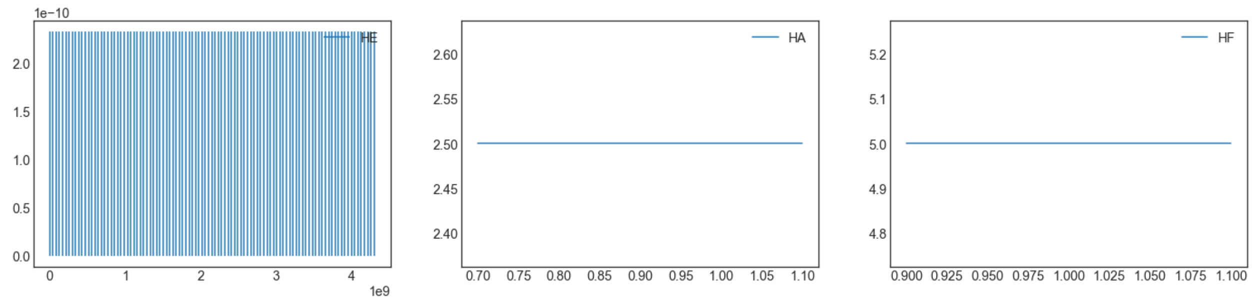
Note that the HE is not a univariate distribution, but for each sample corresponds to the names of the sub-sampled events. However, to simplify the data stream, the HE is saved as the seed for the random number generator that made the sample. Hence, the value of HE is a label for the given sample. If really needed, the exact chosen events can be obtained as follows.

```
import numpy as np
HE = 2618981871 #The random seed (number between 0 and 2**32)
rng = np.random.RandomState(int(HE)) #Initialize a random state with the seed
chosen_ev = list(rng.choice(haz_base.event_name, int(n_ev))) #Obtain the␣
↪corresponding events
```

```
#The first event is
chosen_ev[0]
```

```
'1998209N11335'
```

```
#The values for HE are seeds for the random number generator for the noise sampling␣
↪and
#thus are uniformly sampled numbers between (0, 2**32-1)
haz_iv.plot();
```



The number of events per sub-sample is equal to n_ev

```
#The number of events per sample is equal to n_ev
haz_sub = haz_iv.evaluate(HE=928165924, HI=None, HF = 1.1, HA=None)
#The number for HE is irrelevant, as all samples have the same n_Ev
haz_sub.size - n_ev
```

```
0
```

11.3.3 ImpactFuncSet

The following types of uncertainties can be added:

- MDD: scale the mdd (homogeneously)

  The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_mdd

- PAA: scale the paa (homogeneously)

  The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_paa

- IFi: shift the intensity (homogeneously)

  The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds_int

- IL: sample uniformly from impact function set list

  From the provided list of impact function sets elements are uniformly sampled. For example, impact functions obtained from different calibration methods.

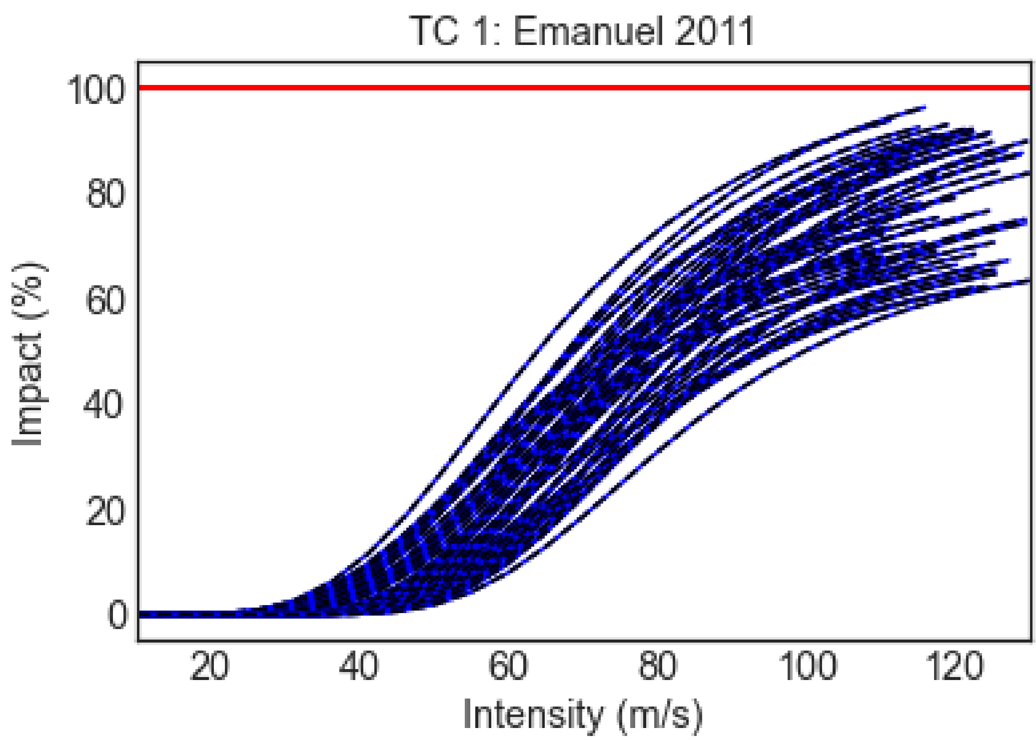
If a bounds is None, this parameter is assumed to have no uncertainty.

```
from climada.entity import ImpactFuncSet, ImpfTropCyclone
impf = ImpfTropCyclone.from_emanuel_usa()
impf_set_base = ImpactFuncSet([impf])
```

It is necessary to specify the hazard type and the impact function id. For simplicity, the default uncertainty input variable only looks at the uncertainty on one single impact function.

```
from climada.engine.unsequa import InputVar
bounds_impfi = [-10, 10] #-10 m/s ; +10m/s uncertainty on the intensity
bounds_mdd = [0.7, 1.1] #-30% - +10% uncertainty on the mdd
bounds_paa = None #No uncertainty in the paa
impf_iv = InputVar.impfset(impf_set_list=[impf_set_base],
                           bounds_impfi=bounds_impfi,
                           bounds_mdd=bounds_mdd,
                           bounds_paa=bounds_paa,
                           haz_id_dict={'TC': [1]})
```

```
#Plot the impact function for 50 random samples (note for the expert, these are not␣
↪global)
n = 50
ax = impf_iv.evaluate().plot()
inten = impf_iv.distr_dict['IFi'].rvs(size=n)
mdd = impf_iv.distr_dict['MDD'].rvs(size=n)
for i, m in zip(inten, mdd):
    impf_iv.evaluate(IFi=i, MDD=m).plot(axis=ax)
ax.get_legend().remove()
```



11.3.4 Entity

The following types of uncertainties can be added:

- DR: value of constant discount rate (homogeneously)

  The value of the discounts in each year is sampled uniformly from a distribution with (min, max) = bounds_disc

- CO: scale the cost (homogeneously)

  The cost of all measures is multiplied by the same number sampled uniformly from a distribution with (min, max) = bounds_cost

- ET: scale the total value (homogeneously)

  The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_totval

- EN: mutliplicative noise (inhomogeneous)

  The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds_noise. EN is the value of the seed for the uniform random number generator.

- EL: sample uniformly from exposure list

  From the provided list of exposure is elements are uniformly sampled. For example, LitPop instances with different exponents.

- MDD: scale the mdd (homogeneously)

  The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_mdd

- PAA: scale the paa (homogeneously)

  The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_paa

- IFi: shift the intensity (homogeneously)

  The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds_int

If a bounds is None, this parameter is assumed to have no uncertainty.

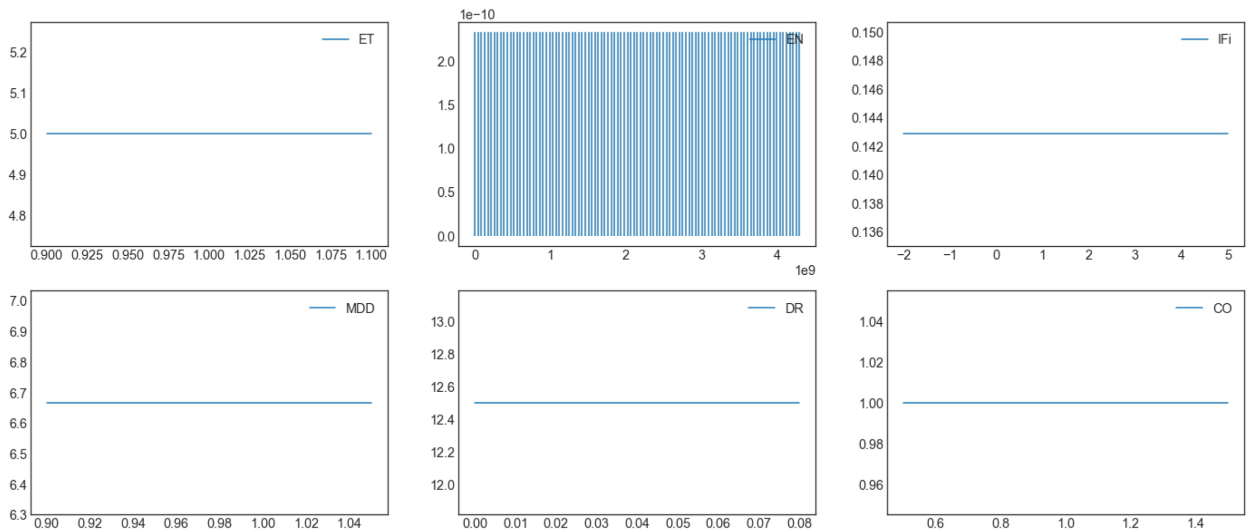
Example: single exposures

```
from climada.entity import Entity
from climada.util.constants import ENT_DEMO_TODAY
ent = Entity.from_excel(ENT_DEMO_TODAY)
ent.exposures.ref_year = 2018
ent.check()
```

```
2022-07-07 15:18:00,292 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:00,293 - climada.entity.exposures.base - INFO - geometry not set.
2022-07-07 15:18:00,294 - climada.entity.exposures.base - INFO - region_id not set.
2022-07-07 15:18:00,294 - climada.entity.exposures.base - INFO - centr_ not set.
```

```python
from climada.engine.unsequa import InputVar
ent_iv = InputVar.ent(
    impf_set_list = [ent.impact_funcs],
    disc_rate = ent.disc_rates,
    exp_list = [ent.exposures],
    meas_set = ent.measures,
    bounds_disc=[0, 0.08],
    bounds_cost=[0.5, 1.5],
    bounds_totval=[0.9, 1.1],
    bounds_noise=[0.3, 1.9],
    bounds_mdd=[0.9, 1.05],
    bounds_paa=None,
    bounds_impfi=[-2, 5],
    haz_id_dict={'TC': [1]}
    )
```

```python
ent_iv.plot();
```



Example: list of Litpop exposures with different exponents

```python
#Define a generic method to make litpop instances with different exponent pairs.
from climada.entity import LitPop
def generate_litpop_base(impf_id, value_unit, haz, assign_centr_kwargs,
                          choice_mn, **litpop_kwargs):
    #In-function imports needed only for parallel computing on Windows
    from climada.entity import LitPop
    litpop_base = []
    for [m, n] in choice_mn:
        print('\n Computing litpop for m=%d, n=%d \n' %(m, n))
        litpop_kwargs['exponents'] = (m, n)
        exp = LitPop.from_countries(**litpop_kwargs)
        exp.gdf['impf_' + haz.haz_type] = impf_id
        exp.gdf.drop('impf_', axis=1, inplace=True)
        if value_unit is not None:
            exp.value_unit = value_unit
        exp.assign_centroids(haz, **assign_centr_kwargs)
```

(continues on next page)

(continued from previous page)

```
        litpop_base.append(exp)
    return litpop_base
```

```
#Define the parameters of the LitPop instances
impf_id = 1
value_unit = None
litpop_kwargs = {
    'countries' : ['CUB'],
    'res_arcsec' : 300,
    'reference_year' : 2020,
}
assign_centr_kwargs={}

# The hazard is needed to assign centroids
from climada.util.constants import HAZ_DEMO_H5
from climada.hazard import Hazard
haz = Hazard.from_hdf5(HAZ_DEMO_H5)
```

```
2022-07-07 15:18:00,956 - climada.hazard.base - INFO - Reading /Users/ckropf/climada/
↪demo/data/tc_fl_1990_2004.h5
```

```
#Generate the LitPop list

choice_mn = [[1, 0.5], [0.5, 1], [1, 1]] #Choice of exponents m,n

litpop_list = generate_litpop_base(impf_id, value_unit, haz, assign_centr_kwargs,␣
↪choice_mn, **litpop_kwargs)
```

```
 Computing litpop for m=1, n=0

2022-07-07 15:18:01,386 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:18:01,968 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:01,997 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,022 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,044 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,069 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,093 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,125 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,187 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,210 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,220 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,221 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:02,233 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,234 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,248 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,249 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,264 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,265 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,278 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,279 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,303 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,330 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,341 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,342 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,351 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,351 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,363 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,364 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,391 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,417 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,443 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,466 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,504 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,528 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,559 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,585 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,596 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,597 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,634 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,666 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,700 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:18:02,734 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,758 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,768 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,769 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,809 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,821 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,822 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,845 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,869 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,896 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,919 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,930 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,959 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-07-07 15:18:04,013 - climada.util.finance - INFO - GDP CUB 2020: 1.074e+11.
2022-07-07 15:18:04,017 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-07-07 15:18:04,028 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:18:04,029 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:04,030 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:18:04,031 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:18:04,032 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:18:04,037 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 2500 centroids.
2022-07-07 15:18:04,039 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:18:04,046 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 78 coordinates.

 Computing litpop for m=0, n=1

2022-07-07 15:18:04,291 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:18:04,812 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:04,842 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:04,869 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:04,894 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:04,923 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:18:04,953 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:04,987 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,049 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,075 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,086 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,087 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,098 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
```

```
2022-07-07 15:18:05,099 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,114 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,115 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,131 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,132 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,148 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,149 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,174 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,199 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,208 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,209 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,220 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,221 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,233 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,234 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,261 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,287 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,310 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,333 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,368 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:05,392 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,420 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,444 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,455 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,456 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,492 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,523 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,557 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,588 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,611 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,622 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,623 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,659 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,671 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,671 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,693 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,716 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,742 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,764 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,775 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,801 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-07-07 15:18:06,617 - climada.util.finance - INFO - GDP CUB 2020: 1.074e+11.
2022-07-07 15:18:06,621 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-07-07 15:18:06,630 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:18:06,631 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:06,631 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:18:06,632 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:18:06,632 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:18:06,636 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 2500 centroids.
2022-07-07 15:18:06,637 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:18:06,643 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 78 coordinates.
```

(continues on next page)

(continued from previous page)

```
 Computing litpop for m=1, n=1

2022-07-07 15:18:06,884 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:18:07,423 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,449 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,473 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,496 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,521 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,544 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,574 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,637 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,660 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,670 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,670 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,682 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,683 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,697 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,698 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,710 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,711 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,723 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,724 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,748 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,773 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,783 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,784 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,793 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,794 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

```
2022-07-07 15:18:07,805 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,806 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,832 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,859 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,882 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,907 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,943 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,968 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,997 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,021 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,032 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:08,032 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,071 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,102 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,136 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,167 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,189 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,200 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:08,201 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,240 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,252 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:08,253 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,276 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,298 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,323 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,345 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,357 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:08,383 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-07-07 15:18:09,253 - climada.util.finance - INFO - GDP CUB 2020: 1.074e+11.
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:09,257 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-07-07 15:18:09,267 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:18:09,268 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:09,268 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:18:09,269 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:18:09,270 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:18:09,275 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 2500 centroids.
2022-07-07 15:18:09,277 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:18:09,283 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 78 coordinates.
```

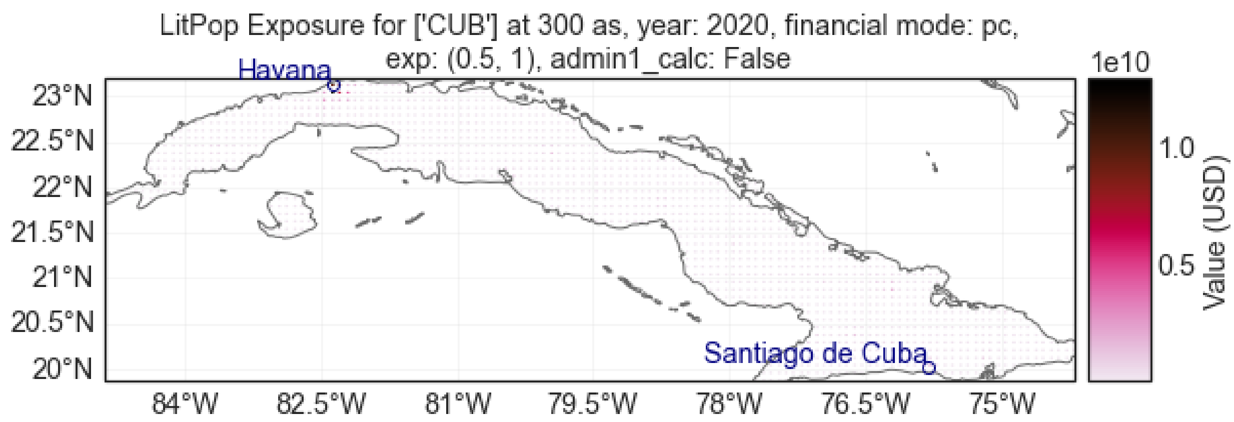
```
from climada.entity import Entity
from climada.util.constants import ENT_DEMO_TODAY
ent = Entity.from_excel(ENT_DEMO_TODAY)
ent.exposures.ref_year = 2020
ent.check()
```

```
2022-07-07 15:18:09,400 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:09,401 - climada.entity.exposures.base - INFO - geometry not set.
2022-07-07 15:18:09,402 - climada.entity.exposures.base - INFO - region_id not set.
2022-07-07 15:18:09,402 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
from climada.engine.unsequa import InputVar
ent_iv = InputVar.ent(
    impf_set_list = [ent.impact_funcs],
    disc_rate = ent.disc_rates,
    exp_list = litpop_list,
    meas_set = ent.measures,
    bounds_disc=[0, 0.08],
    bounds_cost=[0.5, 1.5],
    bounds_totval=[0.9, 1.1],
    bounds_noise=[0.3, 1.9],
    bounds_mdd=[0.9, 1.05],
    bounds_paa=None,
    bounds_impfi=[-2, 5],
    haz_id_dict={'TC': [1]}
    )
```

```
ent_iv.evaluate().exposures.plot_hexbin();
```

```
2022-07-07 15:18:09,448 - climada.util.plot - WARNING - Error parsing coordinate␣
↪system 'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
↪AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0],UNIT[
↪"degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS[
↪"Longitude",EAST],AUTHORITY["EPSG","4326"]]'. Using projection PlateCarree in plot.
```



11.3.5 Entity Future

The following types of uncertainties can be added:

- CO: scale the cost (homogeneously)

  The cost of all measures is multiplied by the same number sampled uniformly from a distribution with (min, max) = bounds_cost

- EG: scale the exposures growth (homogeneously)

  The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_eg

- EN: mutliplicative noise (inhomogeneous)

  The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds_noise. EN is the value of the seed for the uniform random number generator.

- EL: sample uniformly from exposure list

  From the provided list of exposure is elements are uniformly sampled. For example, LitPop instances with different exponents.

- MDD: scale the mdd (homogeneously)

  The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_mdd

- PAA: scale the paa (homogeneously)

  The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_paa

- IFi: shift the impact function intensity (homogeneously)

  The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds_impfi

- IL: sample uniformly from impact function set list

  From the provided list of impact function sets elements are uniformly sampled. For example, impact functions obtained from different calibration methods.

If a bounds is None, this parameter is assumed to have no uncertainty.

Example: single exposures

```
from climada.entity import Entity
from climada.util.constants import ENT_DEMO_FUTURE

ent_fut = Entity.from_excel(ENT_DEMO_FUTURE)
ent_fut.exposures.ref_year = 2040
ent_fut.check()
```

```
2022-07-07 15:18:11,936 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:11,937 - climada.entity.exposures.base - INFO - geometry not set.
2022-07-07 15:18:11,938 - climada.entity.exposures.base - INFO - region_id not set.
2022-07-07 15:18:11,938 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
entfut_iv = InputVar.entfut(
    impf_set_list = [ent_fut.impact_funcs],
    exp_list = [ent_fut.exposures],
    meas_set = ent_fut.measures,
    bounds_cost=[0.6, 1.2],
    bounds_eg=[0.8, 1.5],
    bounds_noise=None,
    bounds_mdd=[0.7, 0.9],
    bounds_paa=[1.3, 2],
    haz_id_dict={'TC': [1]}
    )
```

Example: list of exposures

```
#Define a generic method to make litpop instances with different exponent pairs.
from climada.entity import LitPop
def generate_litpop_base(impf_id, value_unit, haz, assign_centr_kwargs,
                          choice_mn, **litpop_kwargs):
    #In-function imports needed only for parallel computing on Windows
    from climada.entity import LitPop
    litpop_base = []
    for [m, n] in choice_mn:
        print('\n Computing litpop for m=%d, n=%d \n' %(m, n))
        litpop_kwargs['exponents'] = (m, n)
        exp = LitPop.from_countries(**litpop_kwargs)
        exp.gdf['impf_' + haz.haz_type] = impf_id
        exp.gdf.drop('impf_', axis=1, inplace=True)
        if value_unit is not None:
            exp.value_unit = value_unit
        exp.assign_centroids(haz, **assign_centr_kwargs)
        litpop_base.append(exp)
    return litpop_base
```

```
#Define the parameters of the LitPop instances
impf_id = 1
value_unit = None
litpop_kwargs = {
    'countries' : ['CUB'],
    'res_arcsec' : 300,
    'reference_year' : 2040,
```

(continues on next page)

(continued from previous page)

```
}
assign_centr_kwargs={}

# The hazard is needed to assign centroids
from climada.util.constants import HAZ_DEMO_H5
from climada.hazard import Hazard
haz = Hazard.from_hdf5(HAZ_DEMO_H5)
```

```
2022-07-07 15:18:11,958 - climada.hazard.base - INFO - Reading /Users/ckropf/climada/
↪demo/data/tc_fl_1990_2004.h5
```

```
#Generate the LitPop list

choice_mn = [[1, 0.5], [0.5, 1], [1, 1]] #Choice of exponents m,n

litpop_list = generate_litpop_base(impf_id, value_unit, haz, assign_centr_kwargs,␣
↪choice_mn, **litpop_kwargs)
```

```
 Computing litpop for m=1, n=0

2022-07-07 15:18:12,244 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:18:12,773 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,773 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:12,803 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,804 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:12,830 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,831 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:12,854 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,854 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:12,881 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,881 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:12,906 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,907 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:12,937 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,938 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,001 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,002 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,025 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
```

(continues on next page)

(continued from previous page)

```
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,025 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,034 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,035 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,035 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,047 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,048 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,048 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,062 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,063 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,064 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,077 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,078 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,078 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,090 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,090 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,091 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,117 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,118 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,144 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,145 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,155 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,156 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,157 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,165 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,166 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,166 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,178 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,178 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:13,179 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,202 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,203 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,231 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,232 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,256 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,257 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,281 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,281 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,316 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,316 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,344 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,344 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,374 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,375 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,399 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,400 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,412 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,412 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
```

```
2022-07-07 15:18:13,413 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,451 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,452 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,483 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,483 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,521 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,522 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,554 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,555 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:18:13,579 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,579 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,590 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,591 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,592 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,630 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,630 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,643 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,644 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,644 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,666 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,667 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,690 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,691 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,718 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,718 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,742 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,742 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,754 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,780 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-07-07 15:18:14,384 - climada.util.finance - INFO - GDP CUB 2020: 1.074e+11.
2022-07-07 15:18:14,388 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-07-07 15:18:14,396 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:18:14,397 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:14,397 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:18:14,398 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:18:14,398 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:18:14,401 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 2500 centroids.
2022-07-07 15:18:14,403 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:18:14,409 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 78 coordinates.
```

(continues on next page)

(continued from previous page)

```
 Computing litpop for m=0, n=1

2022-07-07 15:18:14,640 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:18:15,172 - climada.entity.exposures.litpop.gpw_population - WARNING -
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,173 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↪Version v4.11
2022-07-07 15:18:15,199 - climada.entity.exposures.litpop.gpw_population - WARNING -
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,200 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↪Version v4.11
2022-07-07 15:18:15,223 - climada.entity.exposures.litpop.gpw_population - WARNING -
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,224 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↪Version v4.11
2022-07-07 15:18:15,249 - climada.entity.exposures.litpop.gpw_population - WARNING -
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,250 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↪Version v4.11
2022-07-07 15:18:15,279 - climada.entity.exposures.litpop.gpw_population - WARNING -
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,279 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↪Version v4.11
2022-07-07 15:18:15,301 - climada.entity.exposures.litpop.gpw_population - WARNING -
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,302 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↪Version v4.11
2022-07-07 15:18:15,330 - climada.entity.exposures.litpop.gpw_population - WARNING -
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,331 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↪Version v4.11
2022-07-07 15:18:15,390 - climada.entity.exposures.litpop.gpw_population - WARNING -
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,391 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↪Version v4.11
2022-07-07 15:18:15,417 - climada.entity.exposures.litpop.gpw_population - WARNING -
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,418 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↪Version v4.11
2022-07-07 15:18:15,428 - climada.entity.exposures.litpop.litpop - INFO - No data
↪point on destination grid within polygon.
2022-07-07 15:18:15,429 - climada.entity.exposures.litpop.gpw_population - WARNING -
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,429 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↪Version v4.11
2022-07-07 15:18:15,441 - climada.entity.exposures.litpop.litpop - INFO - No data
↪point on destination grid within polygon.
2022-07-07 15:18:15,441 - climada.entity.exposures.litpop.gpw_population - WARNING -
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,442 - climada.entity.exposures.litpop.gpw_population - INFO - GPW
↪Version v4.11
2022-07-07 15:18:15,457 - climada.entity.exposures.litpop.litpop - INFO - No data
↪point on destination grid within polygon.
2022-07-07 15:18:15,457 - climada.entity.exposures.litpop.gpw_population - WARNING -
↪Reference year: 2040. Using nearest available year for GPW data: 2020
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:15,458 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

```
2022-07-07 15:18:15,471 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,472 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,472 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,483 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,484 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,485 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,512 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,513 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,539 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,540 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,550 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,551 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,552 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,562 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,562 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,563 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,575 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,575 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,576 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,601 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,602 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,633 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,634 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,661 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,662 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,688 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,689 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:18:15,729 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,730 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,757 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,758 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,787 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,787 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,812 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,813 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,824 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,824 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,825 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,862 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,862 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,893 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,894 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,928 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,929 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,961 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,962 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,984 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,985 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,999 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:16,000 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,000 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,040 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,040 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,051 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:16,052 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:16,052 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,076 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,077 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,103 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,103 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,131 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,132 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,156 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,156 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,167 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
```

```
2022-07-07 15:18:16,195 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-07-07 15:18:17,186 - climada.util.finance - INFO - GDP CUB 2020: 1.074e+11.
2022-07-07 15:18:17,190 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-07-07 15:18:17,200 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:18:17,201 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:17,201 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:18:17,202 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:18:17,203 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:18:17,208 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 2500 centroids.
2022-07-07 15:18:17,210 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:18:17,216 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 78 coordinates.

 Computing litpop for m=1, n=1

2022-07-07 15:18:17,454 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:18:17,967 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:17,967 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:17,997 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:17,998 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,024 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,024 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:18,050 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,050 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,079 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,080 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,111 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,111 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,155 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,155 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,223 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,224 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,250 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,251 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,261 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,262 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,263 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,275 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,275 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,276 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,291 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,291 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,292 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,304 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,305 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,305 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,316 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,317 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,317 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,344 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,345 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:18:18,371 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,372 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,385 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,386 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,387 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,399 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,399 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,400 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,413 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,414 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,414 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,440 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,441 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,468 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,469 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,492 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,493 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,516 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,516 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,553 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
```

```
2022-07-07 15:18:18,553 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,577 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,578 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,607 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,607 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,631 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,632 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:18,644 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,644 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,645 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,681 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,682 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,714 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,714 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,748 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,748 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,782 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,782 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,804 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,804 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,816 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,817 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,818 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,855 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,856 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,866 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,867 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,867 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,891 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,891 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,914 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,914 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,942 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,943 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,965 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,966 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:18:18,978 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:19,006 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-07-07 15:18:19,855 - climada.util.finance - INFO - GDP CUB 2020: 1.074e+11.
2022-07-07 15:18:19,859 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-07-07 15:18:19,869 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:18:19,870 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:19,870 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:18:19,871 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:18:19,872 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:18:19,875 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 2500 centroids.
2022-07-07 15:18:19,878 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:18:19,884 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 78 coordinates.
```

```
from climada.entity import Entity
from climada.util.constants import ENT_DEMO_FUTURE

ent_fut = Entity.from_excel(ENT_DEMO_FUTURE)
ent_fut.exposures.ref_year = 2040
ent_fut.check()
```

```
2022-07-07 15:18:19,989 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:19,989 - climada.entity.exposures.base - INFO - geometry not set.
2022-07-07 15:18:19,990 - climada.entity.exposures.base - INFO - region_id not set.
2022-07-07 15:18:19,990 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
from climada.engine.unsequa import InputVar
entfut_iv = InputVar.entfut(
    impf_set_list = [ent_fut.impact_funcs],
    exp_list = litpop_list,
    meas_set = ent_fut.measures,
    bounds_cost=[0.6, 1.2],
    bounds_eg=[0.8, 1.5],
    bounds_noise=None,
    bounds_mdd=[0.7, 0.9],
    bounds_paa=[1.3, 2],
    haz_id_dict={'TC': [1]}
    )
```

CHAPTER

TWELVE

FORECAST CLASS

This class deals with weather forecasts and uses CLIMADA ImpactCalc.impact() to forecast impacts of weather events on society. It mainly does one thing:

- it contains all plotting and other functionality that are specific for weather forecasts, impact forecasts and warnings

The class is different from the Impact class especially because features of the Impact class like Exceedence frequency curves, annual average impact etc, do not make sense if the hazard is e.g. a 5 day weather forecast. As the class is relatively new, there might be future changes to the datastructure, the methods, and the parameters used to call the methods.

12.1 Example: forecast of building damages due to wind in Switzerland

Before using the forecast class, hazard, exposure and vulnerability need to be created. The hazard looks at the weather forecast from today for an event with two days lead time (meaning the day after tomorrow). generate_WS_forecast_hazard is used to download a current weather forecast for wind gust from opendata.dwd.de. An Impact funtion for building damages due to storms is created. And with only a few lines of code, a LitPop exposure for Switzerland is generated, and the impact is calculated with a default impact function. With a further line of code, the mean damage per grid point for the day after tomorrow is plotted on a map.

```
from datetime import datetime
from cartopy import crs as ccrs

from climada.util.config import CONFIG
from climada.engine.forecast import Forecast
from climada.hazard.storm_europe import StormEurope, generate_WS_forecast_hazard
from climada.entity.impact_funcs.storm_europe import ImpfStormEurope
from climada.entity import ImpactFuncSet
from climada.entity import LitPop
```

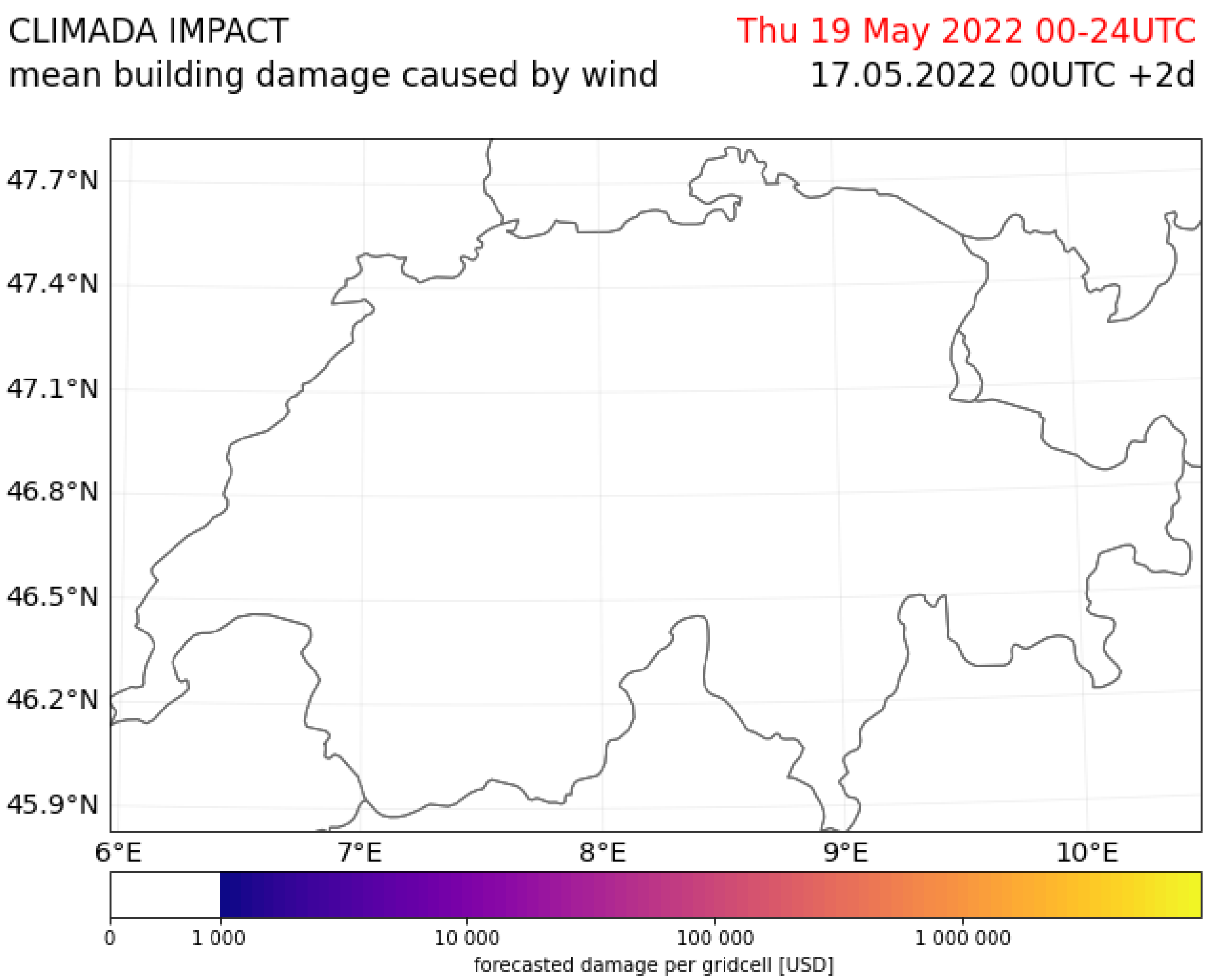
```
#generate hazard
hazard, haz_model, run_datetime, event_date = generate_WS_forecast_hazard()
# generate hazard with forecasts from past dates (works only if the files have␣
↪already been downloaded)
# hazard, haz_model, run_datetime, event_date = generate_WS_forecast_hazard(
#     run_datetime=datetime(2022,5,17),
#     event_date=datetime(2022,5,19))
```

```
#generate vulnerability
impact_function = ImpfStormEurope.from_welker()
impact_function_set = ImpactFuncSet([impact_function])
```

```
#generate exposure and save to file
filename_exp = CONFIG.local_data.save_dir.dir() / ('exp_litpop_Switzerland.hdf5')
if filename_exp.exists():
    exposure = LitPop.from_hdf5(filename_exp)
else:
    exposure = LitPop.from_countries('Switzerland', reference_year=2020)
    exposure.write_hdf5(filename_exp)
```

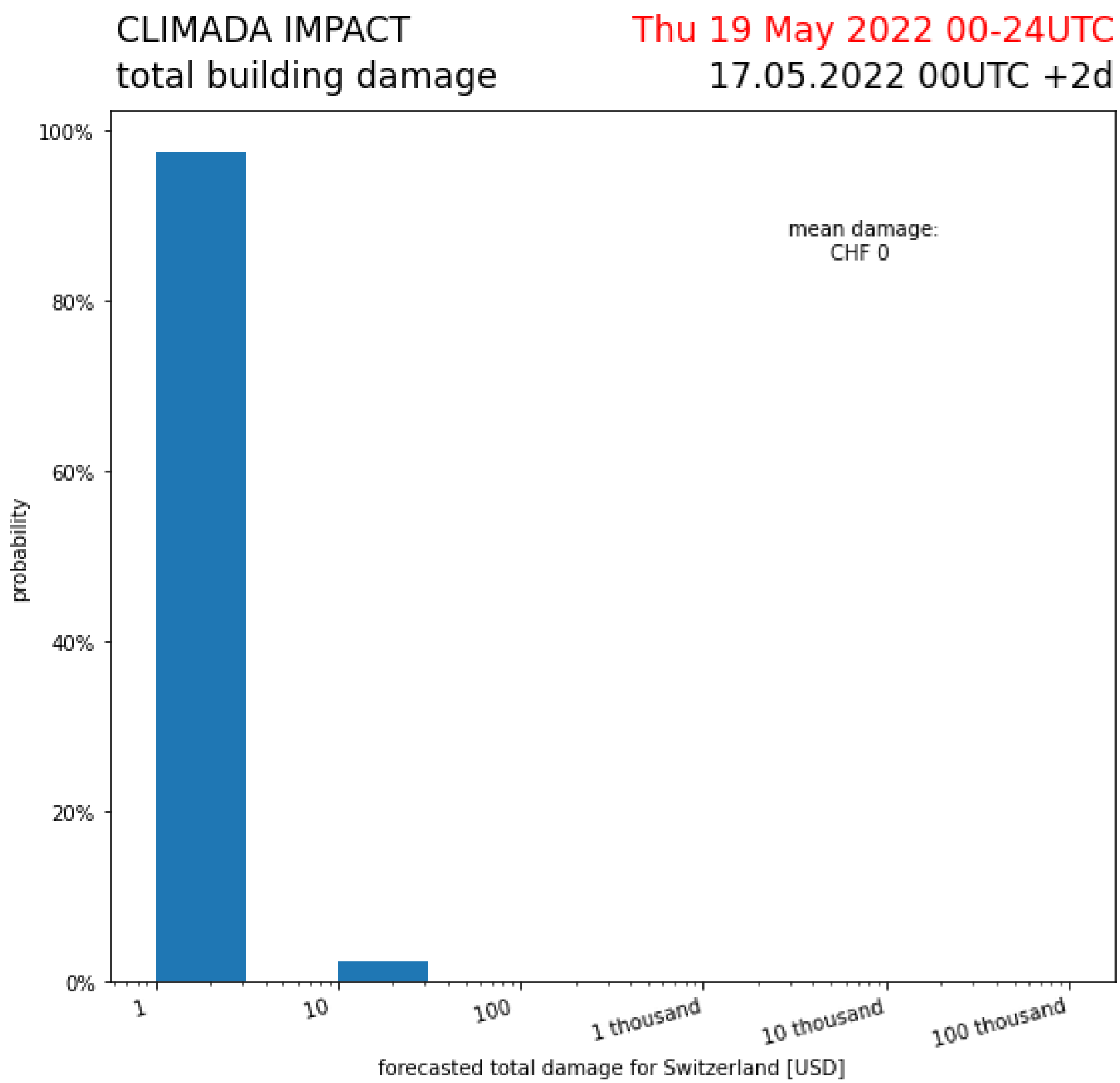
```
#create and calculate Forecast
CH_WS_forecast = Forecast({run_datetime: hazard}, exposure, impact_function_set)
CH_WS_forecast.calc()
```

```
CH_WS_forecast.plot_imp_map(save_fig=False,close_fig=False,proj=ccrs.epsg(2056));
```



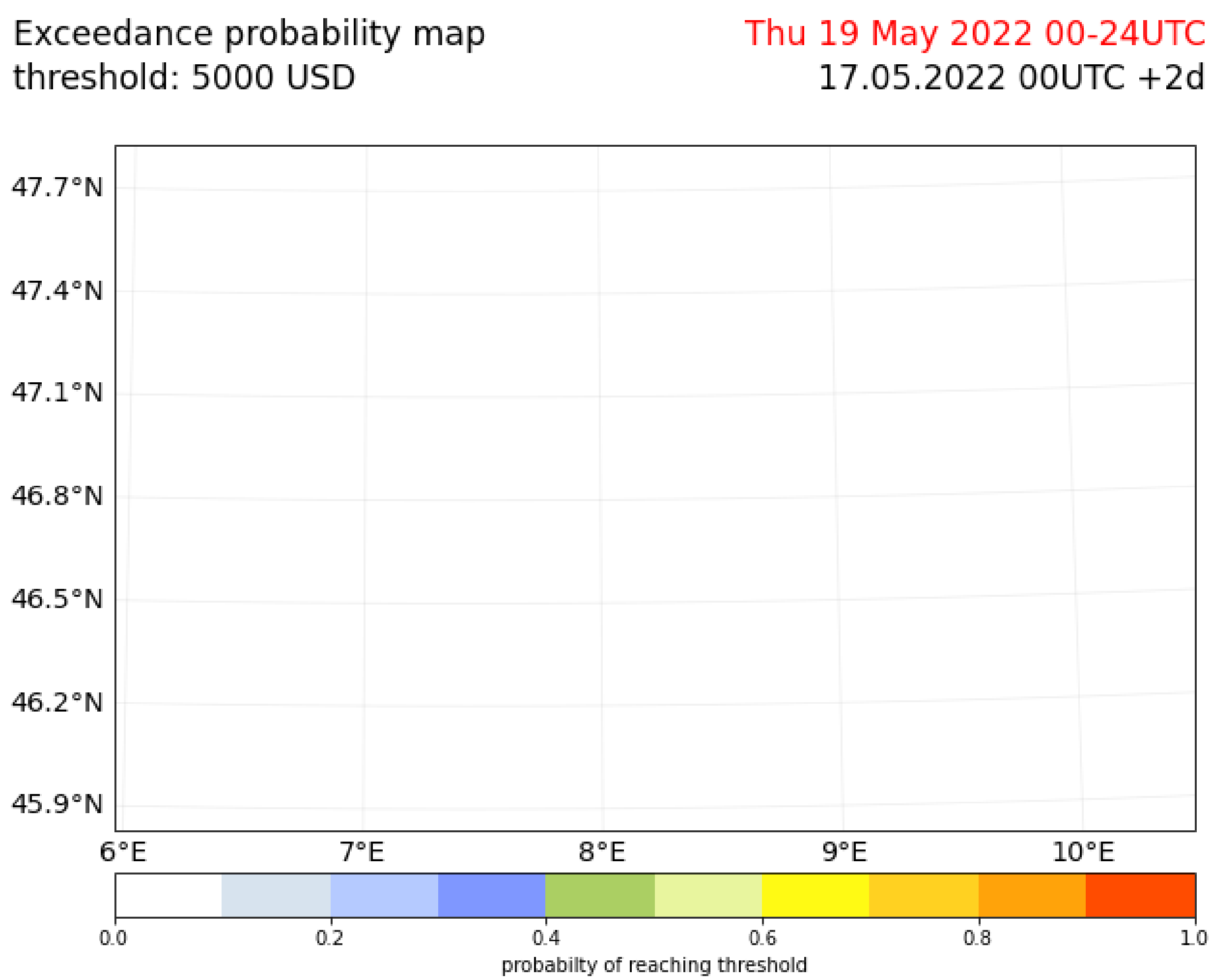
Here you see a different plot highlighting the spread of the impact forecast calculated from the different ensemble members of the weather forecast.

```
CH_WS_forecast.plot_hist(save_fig=False,close_fig=False);
```



It is possible to color the pixels depending on the probability that a certain threshold of impact is reach at a certain grid point

```
CH_WS_forecast.plot_exceedence_prob(threshold=5000, save_fig=False, close_fig=False,
↪proj=ccrs.epsg(2056));
```



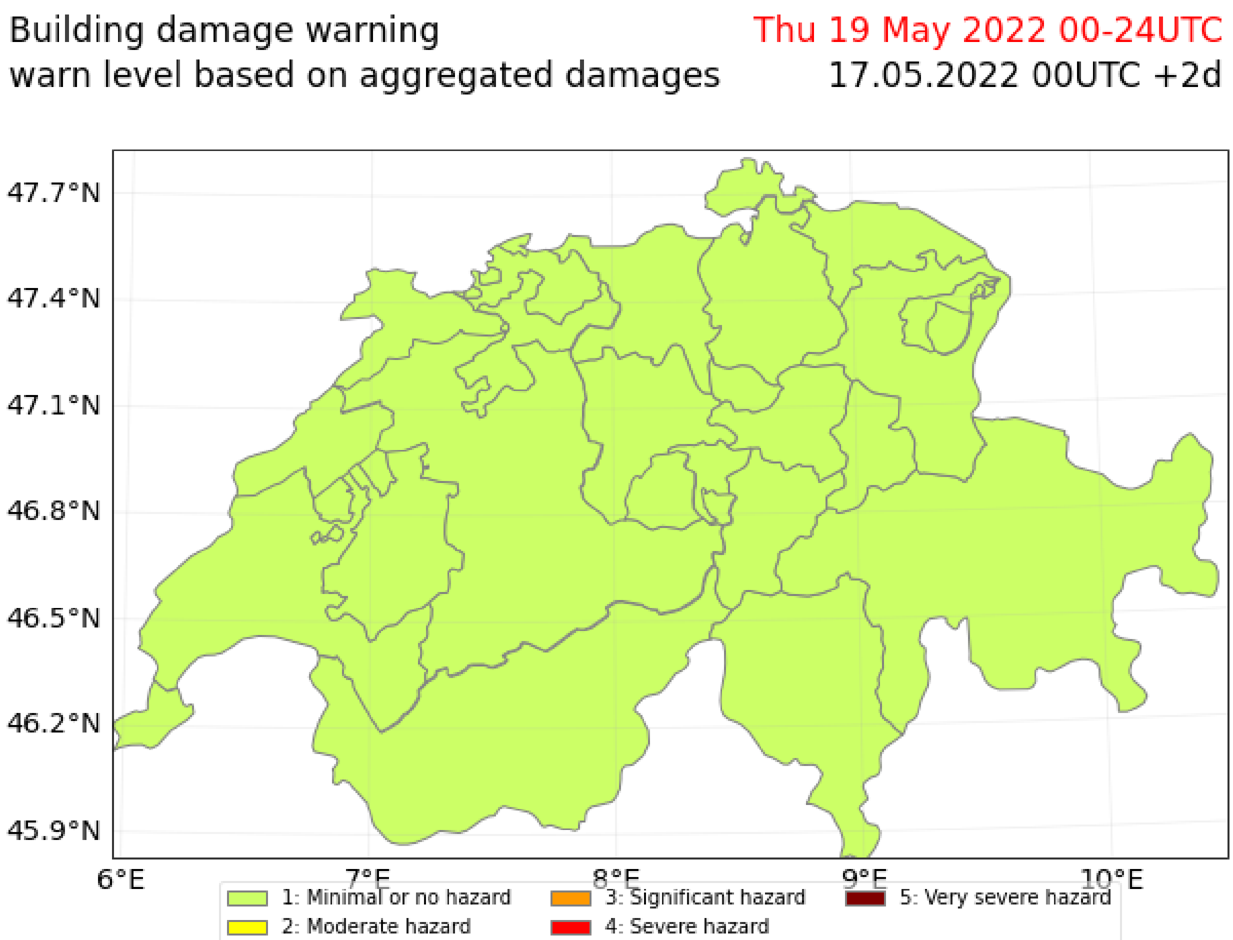
It is possible to color the cantons of Switzerland with warning colors, based on aggregated forecasted impacts in their area.

```
import fiona
from cartopy.io import shapereader
from climada.util.config import CONFIG


#create a file containing the polygons of Swiss cantons using natural earth
cantons_file = CONFIG.local_data.save_dir.dir() / 'cantons.shp'
adm1_shape_file = shapereader.natural_earth(resolution='10m',
                                            category='cultural',
                                            name='admin_1_states_provinces')
if not cantons_file.exists():
    with fiona.open(adm1_shape_file, 'r') as source:
        with fiona.open(
                cantons_file, 'w',
                **source.meta) as sink:

            for f in source:
                if f['properties']['adm0_a3'] == 'CHE':
                    sink.write(f)
CH_WS_forecast.plot_warn_map(str(cantons_file),
```

(continues on next page)

(continued from previous page)

```
                    decision_level = 'polygon',
                    thresholds=[100000,500000,
                                1000000,5000000],
                    probability_aggregation='mean',
                    area_aggregation='sum',
                    title="Building damage warning",
                    explain_text="warn level based on aggregated damages",
                    save_fig=False,
                    close_fig=False,
                    proj=ccrs.epsg(2056));
```



12.2 Example 2: forecast of wind warnings in Switzerland

Instead of a fully fledged socio-economic impact of storms, one can also simplify the hazard, exposure, vulnerability model, by looking at a "neutral" exposure (=1 at every gridpoint) and using a step function as impact function to arrive at warn levels. It also shows how the attributes hazard, exposure or vulnerability can be set before calling calc(), and are then considered in the forecast instead of the defined defaults.

```
from pandas import DataFrame
import numpy as np
```

(continues on next page)

(continued from previous page)

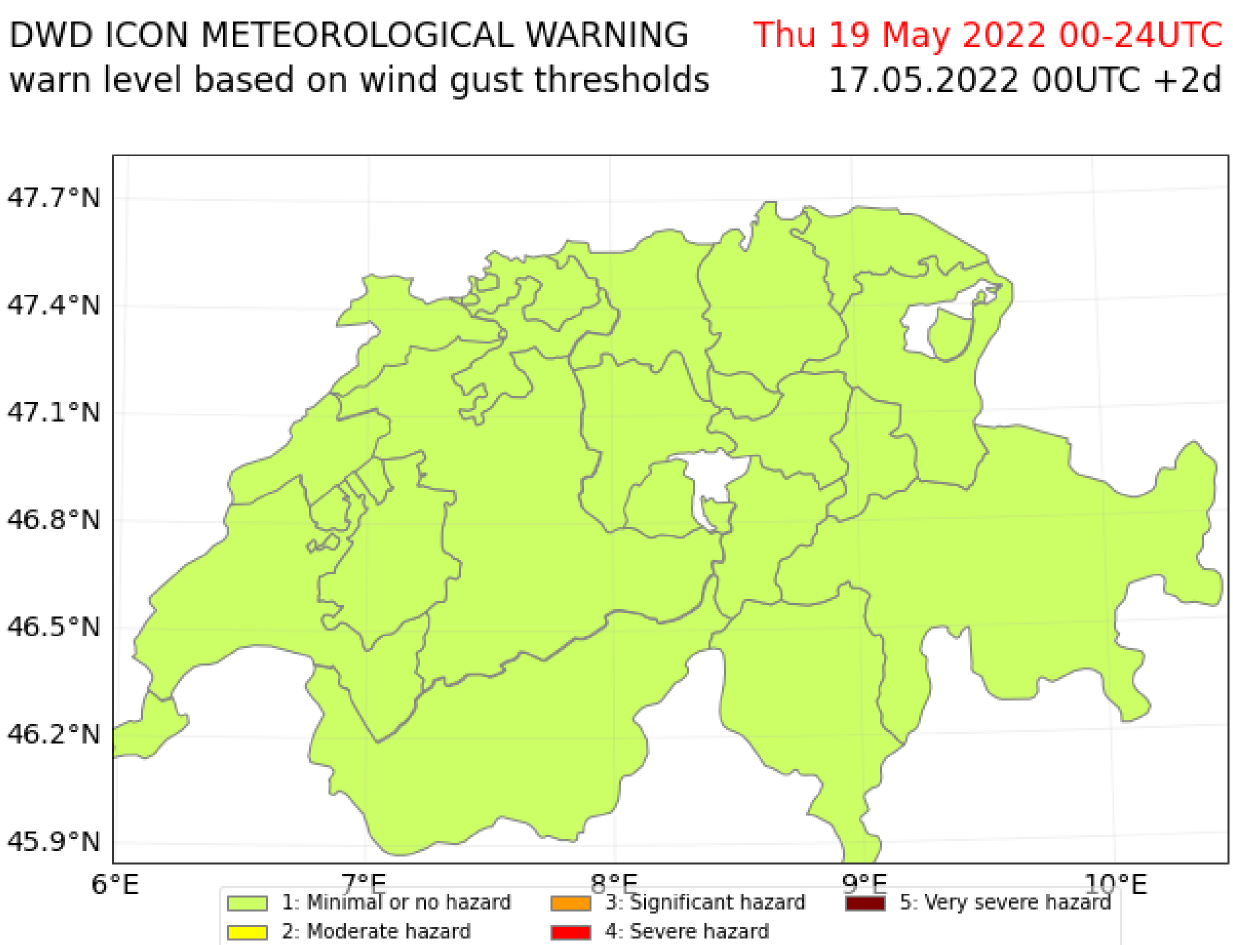
```
from climada.entity.exposures import Exposures
from climada.entity.impact_funcs import ImpactFunc, ImpactFuncSet
import climada.util.plot as u_plot

### generate exposure
# find out which hazard coord to consider
CHE_borders = u_plot._get_borders(np.stack([exposure.gdf.latitude.values,
                                            exposure.gdf.longitude.values],
                                           axis=1)
                                  )
centroid_selection = np.logical_and(np.logical_and(hazard.centroids.lat >= CHE_
↪borders[2],
                                                   hazard.centroids.lat <= CHE_
↪borders[3]),
                                    np.logical_and(hazard.centroids.lon >= CHE_
↪borders[0],
                                                   hazard.centroids.lon <= CHE_
↪borders[1])
                                    )
# Fill DataFrame with values for a "neutral" exposure (value = 1)

exp_df = DataFrame()
exp_df['value'] = np.ones_like(hazard.centroids.lat[centroid_selection]) # provide␣
↪value
exp_df['latitude'] = hazard.centroids.lat[centroid_selection]
exp_df['longitude'] = hazard.centroids.lon[centroid_selection]
exp_df['impf_WS'] = np.ones_like(hazard.centroids.lat[centroid_selection], int)
# Generate Exposures
exp = Exposures(exp_df)
exp.check()
exp.value_unit = 'warn_level'

### generate impact functions
## impact functions for hazard based warnings
haz_type = 'WS'
idx = 1
name = 'warn_level_low_elevation'
intensity_unit = 'm/s'
intensity = np.array([0.0, 19.439,
                      19.44, 24.999,
                      25.0, 30.549,
                      30.55, 38.879,
                      38.88, 100.0])
mdd = np.array([1.0, 1.0,
                2.0, 2.0,
                3.0, 3.0,
                4.0, 4.0,
                5.0, 5.0])
paa = np.ones_like(mdd)
imp_fun_low = ImpactFunc(haz_type, idx, intensity, mdd, paa, intensity_unit, name)
imp_fun_low.check()
# fill ImpactFuncSet
impf_set = ImpactFuncSet([imp_fun_low])
```

```
2022-05-17 09:07:44,196 - climada.entity.impact_funcs.base - WARNING - For intensity␣
↪= 0, mdd != 0 or paa != 0. Consider shifting the origin of the intensity scale. In␣
↪impact.calc the impact is always null at intensity = 0.
```

```
#create and calculate Forecast
warn_forecast = Forecast({run_datetime: hazard}, exp, impf_set)
warn_forecast.calc()
```

The each grid point now has a warnlevel between 1-5 assigned for each event. Now the cantons can be colored based on a threshold on a grid point level. for each warning level it is assessed if 50% of grid points in the area of a canton has at least a 50% probability of reaching the specified threshold.

```
warn_forecast.plot_warn_map(cantons_file,
                            thresholds=[2,3,4,5],
                            decision_level = 'exposure_point',
                            probability_aggregation=0.5,
                            area_aggregation=0.5,
                            title="DWD ICON METEOROLOGICAL WARNING",
                            explain_text="warn level based on wind gust thresholds",
                            save_fig=False,
                            close_fig=False,
                            proj=ccrs.epsg(2056));
```



12.3 Example: Tropical Cylcone

It would be nice to add an example using the tropical cyclone forecasts from the class TCForecast. This has not yet been done.

CHAPTER

THIRTEEN

GOOGLE EARTH ENGINE (GEE) AND IMAGE ANALYSIS

This tutorial explains how to use the module ***climada.util.earth_engine***. It queries data from the Google Earth Engine Python API (https://earthengine.google.com/). A few basic methods of image processing will also be presented using algorythms from Scikit-image (https://scikit-image.org/). A lot of complementary information can be found on this page https://developers.google.com/earth-engine/ (concerns mostly the GEE Java API, but concept and methods are well detailed). GEE is a multi-petabyte catalog of satellite imagery and geospatial datasets. The data are also available on the website of providers, GEE is just more user-friendly as all datasets are available through the same platform.

13.1 Connect to Google Earth Engine API

To access the data, you have to create an account on https://signup.earthengine.google.com/#!/, this step might take some time. Then, install and connect your Python to the API using the terminal. Be sure that climada_env is activated.

In Terminal or Anaconda prompt

$ source activate climada_env

$ conda install -c conda-forge earthengine-api

Then, when the installation is finished, type

$ earthengine authenticate

This will open a web page where you have to enter your account information and a code is provided. Paste it in the terminal.

Then, check in Python if it has worked with the lines below. Import also webbrowser for further steps.

```
import webbrowser

import ee
ee.Initialize()
image = ee.Image('srtm90_v4')
print(image.getInfo())
```


```
{'type': 'Image', 'bands': [{'id': 'elevation', 'data_type': {'type': 'PixelType',
↪'precision': 'int', 'min': -32768, 'max': 32767}, 'dimensions': [432000, 144000],
↪'crs': 'EPSG:4326', 'crs_transform': [0.000833333333333, 0, -180, 0, -0.
↪000833333333333, 60]}], 'version': 1494271934303000.0, 'id': 'srtm90_v4',
↪'properties': {'system:time_start': 950227200000, 'system:time_end': 951177600000,
↪'system:asset_size': 18827626666}}
```

13.2 Obtain images

The module ***climada.util.earth_engine*** enables to select images from some collections of GEE and download them as Geotiff data.

In GEE, you can either access directly one **image** or a **collection**. All products available are detailed on this page https://developers.google.com/earth-engine/datasets/.

```
# Access a specific image
image = ee.Image('LANDSAT/LC08/C01/T1_TOA/LC08_044034_20140318'); #Landsat 8 image,␣
↪with Top of Atmosphere processing, on 2014/03/18

# Access a collection
collection = 'LANDSAT/LE07/C01/T1' #Landsat 7 raw images collection
```

If you have a collection, specification of the time range and area of interest. Then, use methods of the series **obtain_image_type(collection,time_range,area)** depending the type of product needed.

13.2.1 Time range

It depends on the image acquisition period of the targeted satellite and type of images desired (without clouds, from a specific period…)

13.2.2 Area

GEE needs a special format for defining an area of interest. It has to be a GeoJSON Polygon and the coordinates should be first defined in a list and then converted using ee.Geometry. It is possible to use data obtained via Exposure layer. Some examples are given below.

```
#Landsat_composite in Dresden area
area_dresden = list([(13.6, 50.96), (13.9, 50.96), (13.9, 51.12), (13.6, 51.12), (13.
↪6, 50.96)])
area_dresden = ee.Geometry.Polygon(area_dresden)
time_range_dresden = ['2002-07-28', '2002-08-05']

collection_dresden = ('LANDSAT/LE07/C01/T1')
print(type(area_dresden))

#Population density in Switzerland
list_swiss = list([(6.72, 47.88),(6.72, 46.55),(9.72, 46.55),(9.72, 47.88),(6.72, 47.
↪88)])
area_swiss = ee.Geometry.Polygon(list_swiss)
time_range_swiss=['2002-01-01', '2005-12-30']

collection_swiss = ee.ImageCollection('CIESIN/GPWv4/population-density')
print(type(collection_swiss))

#Sentinel 2 cloud-free image in Zürich
collection_zurich = ('COPERNICUS/S2')
list_zurich = list([(8.53, 47.355),(8.55, 47.355),(8.55, 47.376),(8.53, 47.376),(8.53,
↪ 47.355)])
area_zurich = ee.Geometry.Polygon(list_swiss)
time_range_zurich = ['2018-05-01', '2018-07-30']
```

(continues on next page)

(continued from previous page)

```
#Landcover in Europe with CORINE dataset
dataset_landcover = ee.Image('COPERNICUS/CORINE/V18_5_1/100m/2012')
landCover_layer = dataset_landcover.select('landcover')
print(type(landCover_layer))
```

```
<class 'ee.geometry.Geometry'>
<class 'ee.imagecollection.ImageCollection'>
<class 'ee.image.Image'>
```

```
#Methods from climada.util.earth_engine module
def obtain_image_landsat_composite(collection, time_range, area):
    """ Selection of Landsat cloud-free composites in the Earth Engine library
    See also: https://developers.google.com/earth-engine/landsat

    Parameters:
        collection (): name of the collection
        time_range (['YYYY-MT-DY','YYYY-MT-DY']): must be inside the available data
        area (ee.geometry.Geometry): area of interest

    Returns:
        image_composite (ee.image.Image)
     """
    collection = ee.ImageCollection(collection)

    ## Filter by time range and location
    collection_time = collection.filterDate(time_range[0], time_range[1])
    image_area = collection_time.filterBounds(area)
    image_composite = ee.Algorithms.Landsat.simpleComposite(image_area, 75, 3)
    return image_composite

def obtain_image_median(collection, time_range, area):
    """ Selection of median from a collection of images in the Earth Engine library
    See also: https://developers.google.com/earth-engine/reducers_image_collection

    Parameters:
        collection (): name of the collection
        time_range (['YYYY-MT-DY','YYYY-MT-DY']): must be inside the available data
        area (ee.geometry.Geometry): area of interest

    Returns:
        image_median (ee.image.Image)
     """
    collection = ee.ImageCollection(collection)

    ## Filter by time range and location
    collection_time = collection.filterDate(time_range[0], time_range[1])
    image_area = collection_time.filterBounds(area)
    image_median = image_area.median()
    return image_median

def obtain_image_sentinel(collection, time_range, area):
    """ Selection of median, cloud-free image from a collection of images in the␣
↪Sentinel 2 dataset
    See also: https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_
↪S2
```

(continues on next page)

(continued from previous page)

```
    Parameters:
        collection (): name of the collection
        time_range (['YYYY-MT-DY','YYYY-MT-DY']): must be inside the available data
        area (ee.geometry.Geometry): area of interest

    Returns:
        sentinel_median (ee.image.Image)
     """
#First, method to remove cloud from the image
    def maskclouds(image):
        band_qa = image.select('QA60')
        cloud_mask = ee.Number(2).pow(10).int()
        cirrus_mask = ee.Number(2).pow(11).int()
        mask = band_qa.bitwiseAnd(cloud_mask).eq(0) and(
            band_qa.bitwiseAnd(cirrus_mask).eq(0))
        return image.updateMask(mask).divide(10000)

    sentinel_filtered = (ee.ImageCollection(collection).
                         filterBounds(area).
                         filterDate(time_range[0], time_range[1]).
                         filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20)).
                         map(maskclouds))

    sentinel_median = sentinel_filtered.median()
    return sentinel_median
```

```
#Application to examples
composite_dresden = obtain_image_landsat_composite(collection_dresden, time_range_
↪dresden, area_dresden)
median_swiss = obtain_image_median(collection_swiss, time_range_swiss, area_swiss)
zurich_median = obtain_image_sentinel(collection_zurich, time_range_zurich, area_
↪zurich)

#Selection of specific bands from an image
zurich_band = zurich_median.select(['B4','B3','B2'])


print(composite_dresden.getInfo())
print(type(median_swiss))
print(type(zurich_band))
```


```
{'type': 'Image', 'bands': [{'id': 'B1', 'data_type': {'type': 'PixelType', 'precision
↪': 'int', 'min': 0, 'max': 255}, 'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0,␣
↪1, 0]}, {'id': 'B2', 'data_type': {'type': 'PixelType', 'precision': 'int', 'min':␣
↪0, 'max': 255}, 'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0, 1, 0]}, {'id': 'B3
↪', 'data_type': {'type': 'PixelType', 'precision': 'int', 'min': 0, 'max': 255},
↪'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0, 1, 0]}, {'id': 'B4', 'data_type':
↪{'type': 'PixelType', 'precision': 'int', 'min': 0, 'max': 255}, 'crs': 'EPSG:4326',
↪ 'crs_transform': [1, 0, 0, 0, 1, 0]}, {'id': 'B5', 'data_type': {'type': 'PixelType
↪', 'precision': 'int', 'min': 0, 'max': 255}, 'crs': 'EPSG:4326', 'crs_transform':␣
↪[1, 0, 0, 0, 1, 0]}, {'id': 'B6_VCID_1', 'data_type': {'type': 'PixelType',
↪'precision': 'int', 'min': 0, 'max': 255}, 'crs': 'EPSG:4326', 'crs_transform': [1,␣
↪0, 0, 0, 1, 0]}, {'id': 'B6_VCID_2', 'data_type': {'type': 'PixelType', 'precision
↪': 'int', 'min': 0, 'max': 255}, 'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0,␣
↪1, 0]}, {'id': 'B7', 'data_type': {'type': 'PixelType', 'precision': 'int', 'min':␣
```


(continues on next page)

(continued from previous page)

```
↪0, 'max': 255}, 'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0, 1, 0]}, {'id': 'B8
↪', 'data_type': {'type': 'PixelType', 'precision': 'int', 'min': 0, 'max': 255},
↪'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0, 1, 0]}]}
<class 'ee.image.Image'>
<class 'ee.image.Image'>
```

13.3 Download images

To visualize and work on images, it is easier to download them (in Geotiff), using the **get_url(name, image, scale, region)** method. The image will be downloaded regarding a region and a scale. 'region' is obtained from the area, but the format has to be adjusted using **get_region(geom)** method.

```
def get_region(geom):
    """Get the region of a given geometry, needed for exporting tasks.

    Parameters:
        geom (ee.Geometry, ee.Feature, ee.Image): region of interest

    Returns:
        region (list)
    """
    if isinstance(geom, ee.Geometry):
        region = geom.getInfo()["coordinates"]
    elif isinstance(geom, (ee.Feature, ee.Image)):
        region = geom.geometry().getInfo()["coordinates"]
    return region


region_dresden = get_region(area_dresden)
region_swiss = get_region(area_swiss)
region_zurich= get_region(area_zurich)
```

```
# If you want to apply this function to a list of regions:
region_list = [get_region(geom) for geom in [area_dresden, area_zurich]]
```

```
def get_url(name, image, scale, region):
    """It will open and download automatically a zip folder containing Geotiff data␣
↪of 'image'.
    If additional parameters are needed, see also:
    https://github.com/google/earthengine-api/blob/master/python/ee/image.py

    Parameters:
        name (str): name of the created folder
        image (ee.image.Image): image to export
        scale (int): resolution of export in meters (e.g: 30 for Landsat)
        region (list): region of interest

    Returns:
        path (str)
     """
    path = image.getDownloadURL({
        'name':(name),
        'scale': scale,
```

(continues on next page)

(continued from previous page)

```
        'region':(region)
        })

    webbrowser.open_new_tab(path)
    return path



url_swiss = get_url('swiss_pop', median_swiss, 900, region_swiss)
url_dresden = get_url('dresden', composite_dresden, 30, region_dresden)
url_landcover = get_url('landcover_swiss', landCover_layer, 100, region_swiss)

#For the example of Zürich, due to size, it doesn't work on Jupyter Notebook but it␣
↪works on Python
#url_zurich = get_url('sentinel', zurich_band, 10, region_zurich)

print(url_swiss)
print(url_dresden)
print(url_landcover)
```

```
https://earthengine.googleapis.com/api/download?
↪docid=6b6d96f567d6a055188c8c17dd24bcb8&token=00a796601efe425c821777a284bff361
https://earthengine.googleapis.com/api/download?
↪docid=15182f82ba65ce24f62305e4465ac21c&token=5da59a20bb84d79bcf7ce958855fe848
https://earthengine.googleapis.com/api/download?
↪docid=07c14e22d96a33fc72a7ba16c2178a6a&token=0cfa0cd6537257e96600d10647375ff4
```

13.4 Image Visualization and Processing

In this section, basics methods of image processing will be presented as well as tools to visualize the image. The images downloaded before are used as examples but these methods works with all tif data. Scikit-image (https://scikit-image.org/) needs to be imported.

First, bands can be combined, for example to obtain an RGB image from the Red, Blue and Green bands. It is done with gdal_merge.py (see: https://gdal.org/programs/gdal_merge.html). It is better if bands are named just as B1, B2, B3 … in the folder containing the image data.

If you don't have any bands that you want to combine, you don't have to execute the following codes for this tutorial.

In Terminal or Anaconda prompt (be sure that climada_env is activated):

$ cd '/your/path/to/image_downloaded_folder'

$ gdal_merge.py -separate -co PHOTOMETRIC=RGB -o merged.tif B_red.tif B_blue.tif B_green.tif

The RGB image will be merged.tif

```
import numpy as np
from skimage import data
import matplotlib.pyplot as plt
from skimage.color import rgb2gray

from skimage.io import imread
from skimage import exposure
from skimage.filters import try_all_threshold
```

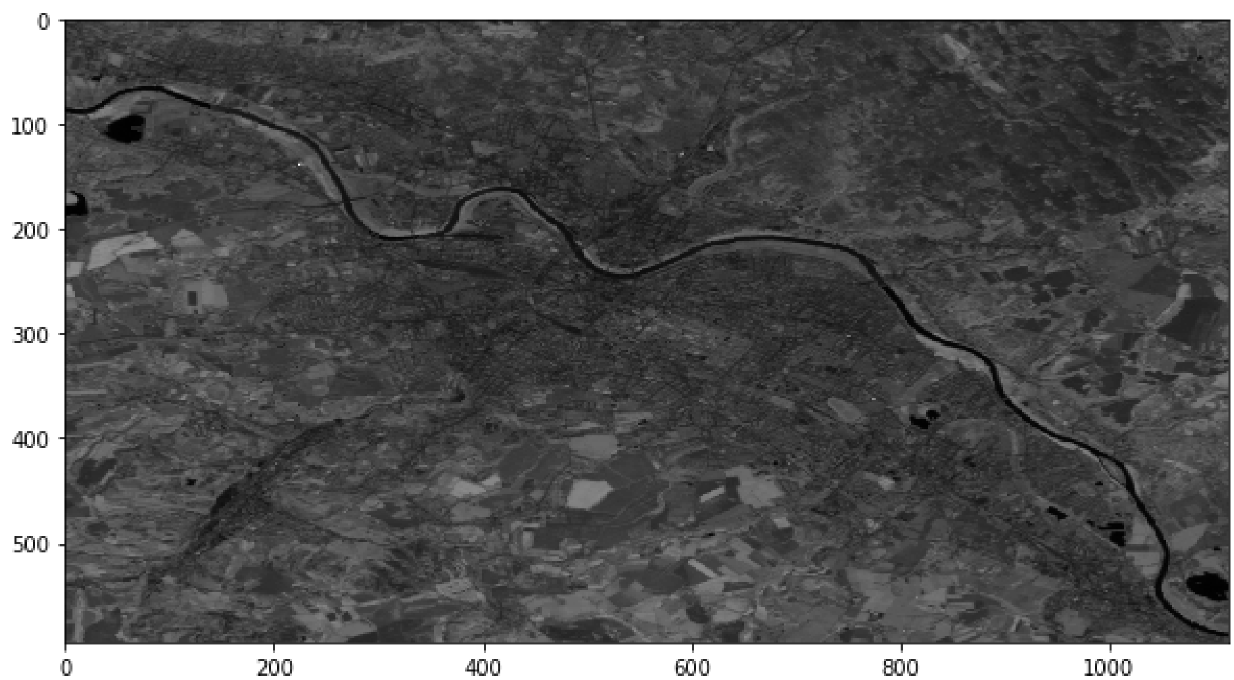
(continues on next page)

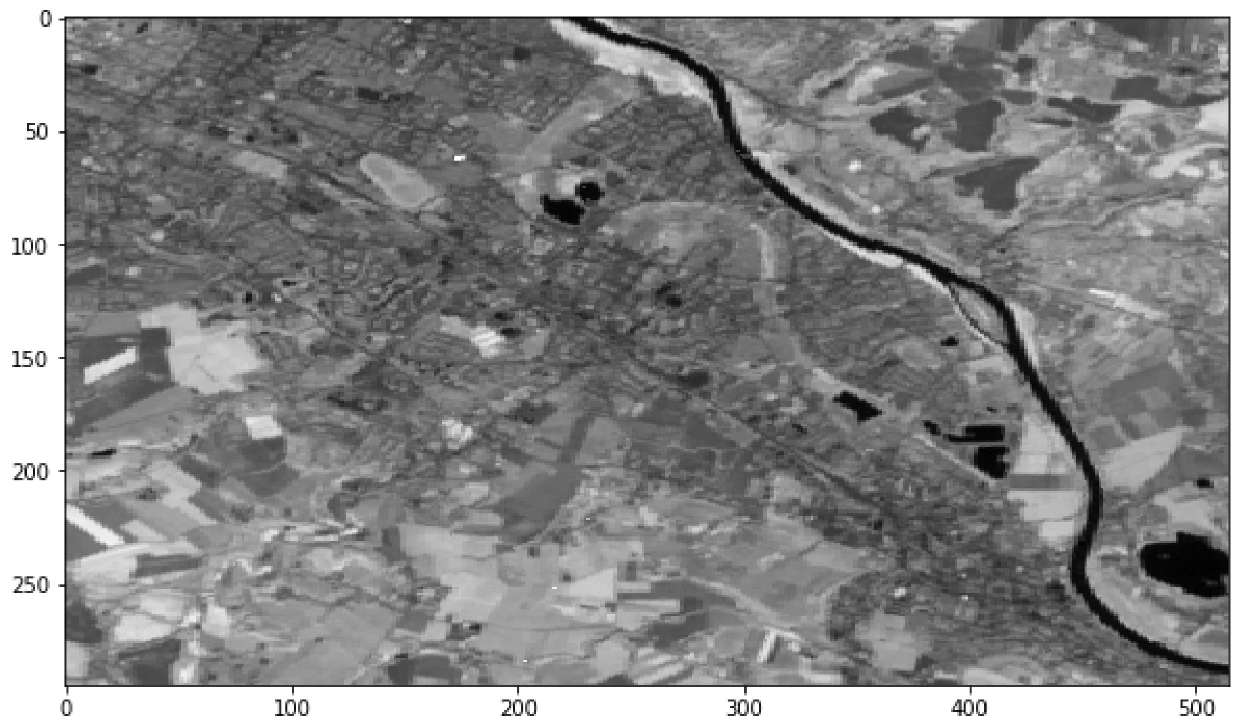
(continued from previous page)

```
from skimage.filters import threshold_otsu, threshold_local
from skimage import measure
from skimage import feature
```

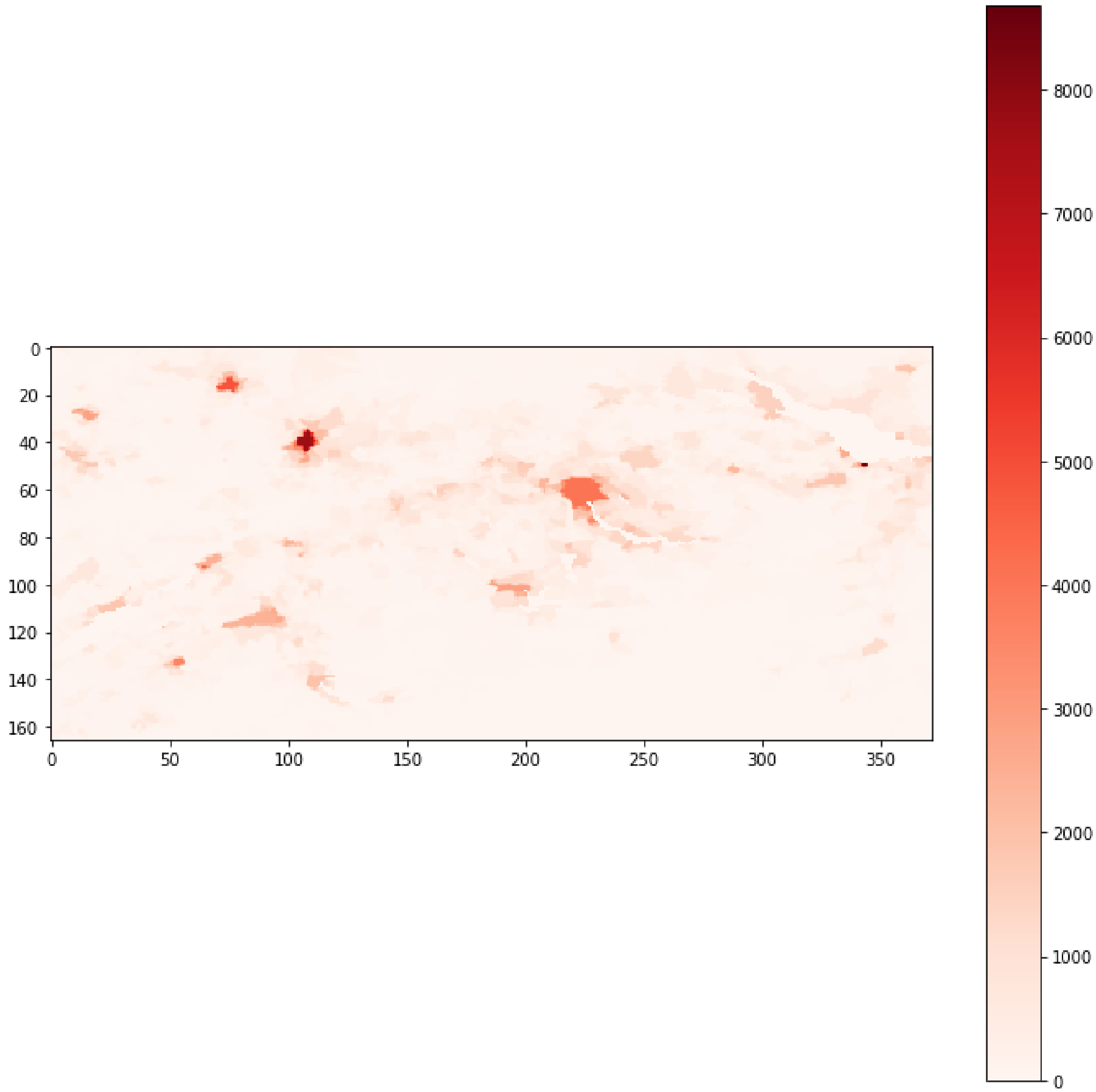
```
from climada.util import DEMO_DIR

swiss_pop = DEMO_DIR.joinpath('earth_engine', 'population-density_median.tif')
dresden = DEMO_DIR.joinpath('earth_engine', 'dresden.tif') #B4 of Dresden example
```

```
#Read a tif in python and Visualize the image
image_dresden = imread(dresden)
plt.figure(figsize=(10, 10))
plt.imshow(image_dresden, cmap='gray', interpolation='nearest')
plt.axis()
plt.show()

#Crop the image
image_dresden_crop=image_dresden[300:700,600:1400]
plt.figure(figsize=(10, 10))
plt.imshow(image_dresden_crop, cmap='gray', interpolation='nearest')
plt.axis()
plt.show()
```






```
image_pop= imread(swiss_pop)
plt.figure(figsize=(12, 12))
plt.imshow(image_pop, cmap='Reds', interpolation='nearest')
plt.colorbar()
plt.axis()
plt.show()
```

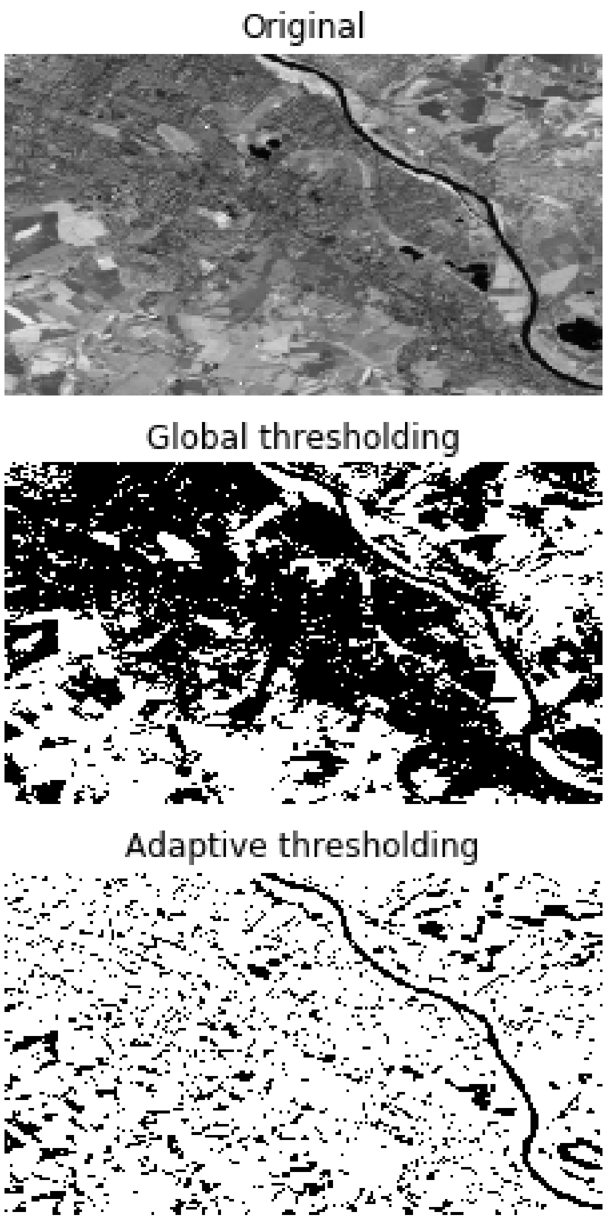



```
#Thresholding: Selection of pixels with regards with their value

global_thresh = threshold_otsu(image_dresden_crop)
binary_global = image_dresden_crop > global_thresh

block_size = 35
adaptive_thresh = threshold_local(image_dresden_crop, block_size, offset=10)
binary_adaptive = image_dresden_crop > adaptive_thresh

fig, axes = plt.subplots(nrows=3, figsize=(7, 8))
ax = axes.ravel()
plt.gray()

ax[0].imshow(image_dresden_crop)
```

(continues on next page)

(continued from previous page)

```
ax[0].set_title('Original')

ax[1].imshow(binary_global)
ax[1].set_title('Global thresholding')

ax[2].imshow(binary_adaptive)
ax[2].set_title('Adaptive thresholding')

for a in ax:
    a.axis('off')
plt.show()

print(np.sum(binary_global))
```



```
64832
```

CHAPTER

FOURTEEN

DATA API

This tutorial is separated into three main parts: the first two parts shows how to find and get data to do impact calculations and should be enough for most users. The third part provides more detailed information on how the API is built.

14.1 Contents

- *Finding Datasets*
  - *Data types and data type groups*
  - *Datasets and Properties*
- *Basic impact calculation*
  - *Wrapper functions to open datasets as CLIMADA objects*
  - *Calculate the impact*
- *Technical Information*
  - *Server*
  - *Client*
  - *Metadata*
  - *Download*

14.2 Finding datasets

```
from climada.util.api_client import Client
client = Client()
```

14.2.1 Data types and data type groups

The datasets are first separated into 'data_type_groups', which represent the main classes of CLIMADA (exposures, hazard, vulnerability, …). So far, data is available for exposures and hazard. Then, data is separated into data_types, representing the different hazards and exposures available in CLIMADA

```
import pandas as pd
data_types = client.list_data_type_infos()

dtf = pd.DataFrame(data_types)
dtf.sort_values(['data_type_group', 'data_type'])
```

```
          data_type data_type_group  status description  \
3   crop_production       exposures  active        None
0            litpop       exposures  active        None
5         centroids          hazard  active        None
2       river_flood          hazard  active        None
4      storm_europe          hazard  active        None
1  tropical_cyclone          hazard  active        None

                                          properties
3  [{'property': 'crop', 'mandatory': True, 'desc...
0  [{'property': 'res_arcsec', 'mandatory': False...
5                                                 []
2  [{'property': 'res_arcsec', 'mandatory': False...
4  [{'property': 'country_iso3alpha', 'mandatory'...
1  [{'property': 'res_arcsec', 'mandatory': True,...
```

14.2.2 Datasets and Properties

For each data type, the single datasets can be differentiated based on properties. The following function provides a table listing the properties and possible values. This table does not provide information on properties that can be combined but the search can be refined in order to find properties to query a unique dataset. Note that a maximum of 10 property values are shown here, but many more countries are available for example.

```
litpop_dataset_infos = client.list_dataset_infos(data_type='litpop')
```

```
all_properties = client.get_property_values(litpop_dataset_infos)
```

```
all_properties.keys()
```

```
dict_keys(['res_arcsec', 'exponents', 'fin_mode', 'spatial_coverage', 'country_
↪iso3alpha', 'country_name', 'country_iso3num'])
```

Refining the search:

```
# as datasets are usually available per country, chosing a country or global dataset␣
↪reduces the options
# here we want to see which datasets are available for litpop globally:
client.get_property_values(litpop_dataset_infos, known_property_values = {'spatial_
↪coverage':'global'})
```

```
{'res_arcsec': ['150'],
 'exponents': ['(0,1)', '(1,1)', '(3,0)'],
 'fin_mode': ['pop', 'pc'],
 'spatial_coverage': ['global']}
```

```
#and here for Switzerland:
client.get_property_values(litpop_dataset_infos, known_property_values = {'country_
↪name':'Switzerland'})
```

```
{'res_arcsec': ['150'],
 'exponents': ['(3,0)', '(0,1)', '(1,1)'],
 'fin_mode': ['pc', 'pop'],
 'spatial_coverage': ['country'],
 'country_iso3alpha': ['CHE'],
 'country_name': ['Switzerland'],
 'country_iso3num': ['756']}
```

14.3 Basic impact calculation

We here show how to make a basic impact calculation with tropical cyclones for Haiti, for the year 2040, rcp4.5 and generated with 10 synthetic tracks. For more technical details on the API, see below.

14.3.1 Wrapper functions to open datasets as CLIMADA objects

The wrapper functions client.get_hazard()

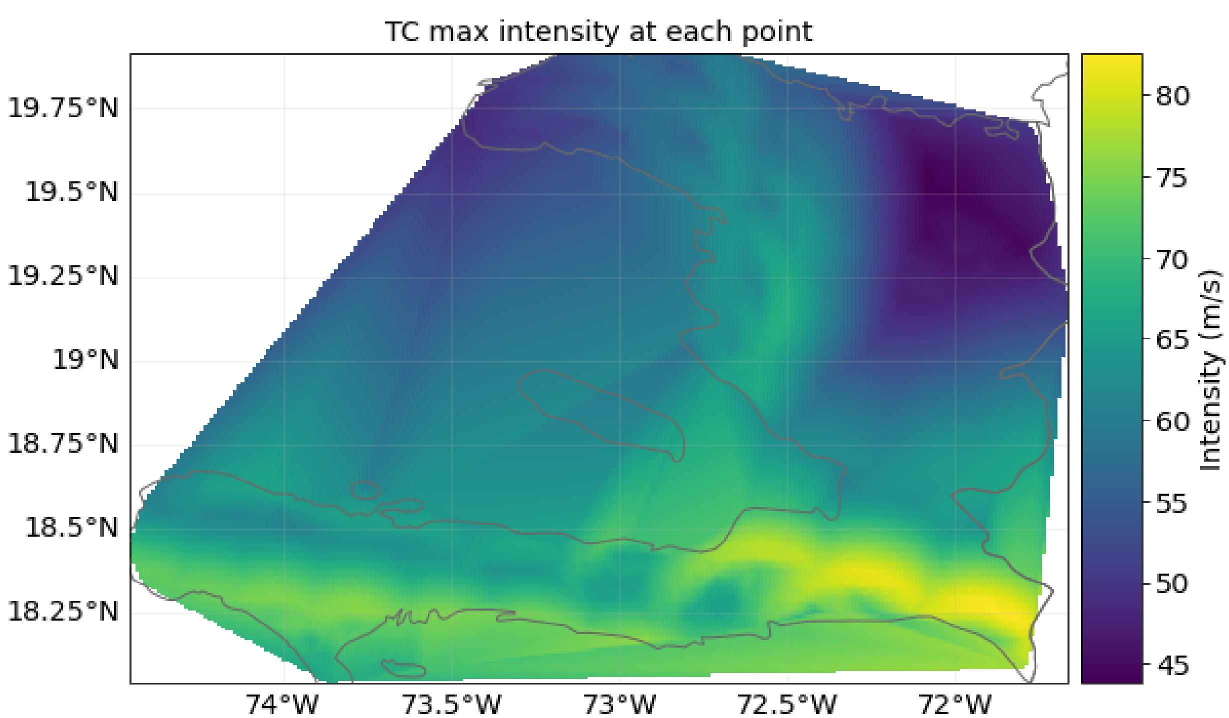
gets the dataset information, downloads the data and opens it as a hazard instance

```
tc_dataset_infos = client.list_dataset_infos(data_type='tropical_cyclone')
client.get_property_values(tc_dataset_infos, known_property_values = {'country_name':
↪'Haiti'})
```

```
{'res_arcsec': ['150'],
 'climate_scenario': ['rcp26', 'rcp45', 'rcp85', 'historical', 'rcp60'],
 'ref_year': ['2040', '2060', '2080'],
 'nb_synth_tracks': ['50', '10'],
 'spatial_coverage': ['country'],
 'tracks_year_range': ['1980_2020'],
 'country_iso3alpha': ['HTI'],
 'country_name': ['Haiti'],
 'country_iso3num': ['332'],
 'resolution': ['150 arcsec']}
```

```
client = Client()
tc_haiti = client.get_hazard('tropical_cyclone', properties={'country_name': 'Haiti',
↪'climate_scenario': 'rcp45', 'ref_year':'2040', 'nb_synth_tracks':'10'})
tc_haiti.plot_intensity(0);
```

```
https://climada.ethz.ch/data-api/v1/dataset        climate_
↪scenario=rcp45        country_name=Haiti        data_type=tropical_
↪cyclone        limit=100000        name=None        nb_synth_tracks=10        ref_
↪year=2040        status=active        version=None
2022-07-01 15:55:23,593 - climada.util.api_client - WARNING - Download failed: /Users/
↪szelie/climada/data/hazard/tropical_cyclone/tropical_cyclone_10synth_tracks_
↪150arcsec_rcp45_HTI_2040/v1/tropical_cyclone_10synth_tracks_150arcsec_rcp45_HTI_
↪2040.hdf5 has the wrong size:8189651 instead of 7781902, retrying...
2022-07-01 15:55:26,786 - climada.hazard.base - INFO - Reading /Users/szelie/climada/
↪data/hazard/tropical_cyclone/tropical_cyclone_10synth_tracks_150arcsec_rcp45_HTI_
↪2040/v1/tropical_cyclone_10synth_tracks_150arcsec_rcp45_HTI_2040.hdf5
2022-07-01 15:55:27,129 - climada.util.plot - WARNING - Error parsing coordinate␣
↪system 'GEOGCRS["WGS 84",ENSEMBLE["World Geodetic System 1984 ensemble",MEMBER[
↪"World Geodetic System 1984 (Transit)"],MEMBER["World Geodetic System 1984 (G730)"],
↪MEMBER["World Geodetic System 1984 (G873)"],MEMBER["World Geodetic System 1984␣
↪(G1150)"],MEMBER["World Geodetic System 1984 (G1674)"],MEMBER["World Geodetic␣
↪System 1984 (G1762)"],ELLIPSOID["WGS 84",6378137,298.257223563,LENGTHUNIT["metre",
↪1]],ENSEMBLEACCURACY[2.0]],PRIMEM["Greenwich",0,ANGLEUNIT["degree",0.
↪0174532925199433]],CS[ellipsoidal,2],AXIS["geodetic latitude (Lat)",north,ORDER[1],
↪ANGLEUNIT["degree",0.0174532925199433]],AXIS["geodetic longitude (Lon)",east,
↪ORDER[2],ANGLEUNIT["degree",0.0174532925199433]],USAGE[SCOPE["Horizontal component␣
↪of 3D system."],AREA["World."],BBOX[-90,-180,90,180]],ID["EPSG",4326]]'. Using␣
↪projection PlateCarree in plot.
```



The wrapper functions client.get_litpop()

gets the default litpop, with exponents (1,1) and 'produced capital' as financial mode. If no country is given, the global dataset will be downloaded.

```
litpop_default = client.get_property_values(litpop_dataset_infos, known_property_
↪values = {'fin_mode':'pc', 'exponents':'(1,1)'})
```

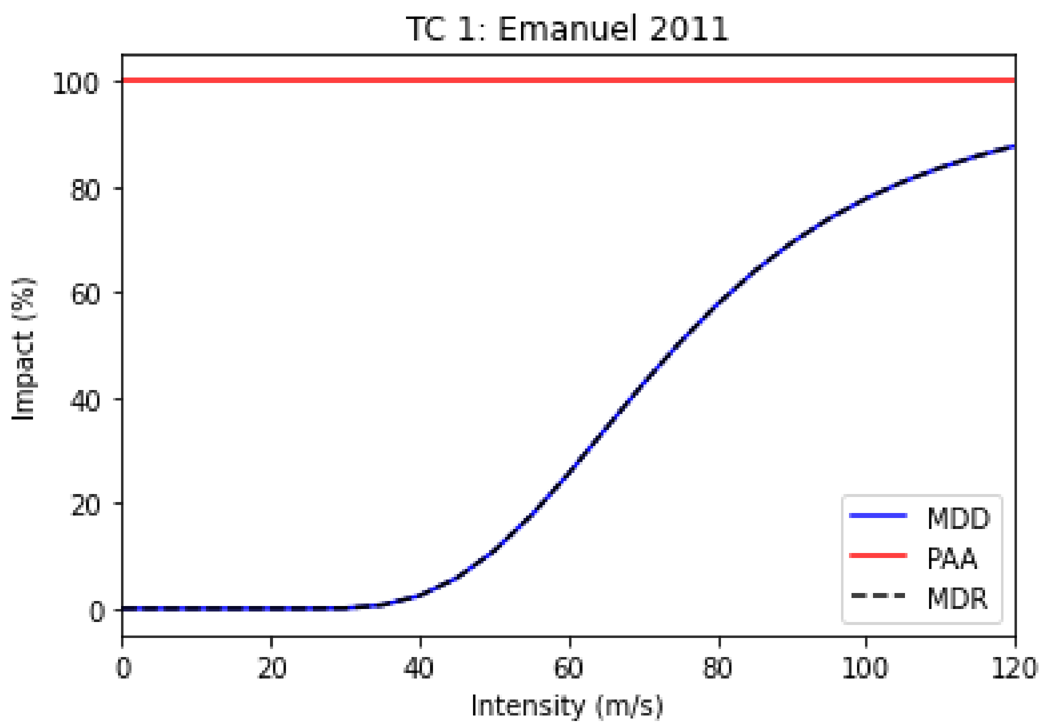
```
litpop = client.get_litpop(country='Haiti')
```

```
https://climada.ethz.ch/data-api/v1/dataset        country_name=Haiti        data_
↪type=litpop        exponents=(1,
↪1)        limit=100000        name=None        status=active        version=None
2022-07-01 15:55:31,047 - climada.entity.exposures.base - INFO - Reading /Users/
↪szelie/climada/data/exposures/litpop/LitPop_150arcsec_HTI/v1/LitPop_150arcsec_HTI.
↪hdf5
```

Get the default impact function for tropical cyclones

```
from climada.entity.impact_funcs import ImpactFuncSet, ImpfTropCyclone

imp_fun = ImpfTropCyclone.from_emanuel_usa()
imp_fun.check()
imp_fun.plot()

imp_fun_set = ImpactFuncSet([imp_fun])

litpop.impact_funcs = imp_fun_set
```

```
2022-01-31 22:30:21,359 - climada.entity.impact_funcs.base - WARNING - For intensity␣
↪= 0, mdd != 0 or paa != 0. Consider shifting the origin of the intensity scale. In␣
↪impact.calc the impact is always null at intensity = 0.
```



14.3.2 Calculate the impact

```
from climada.engine import ImpactCalc
impact = ImpactCalc(litpop, imp_fun_set, tc_haiti).impact()
```

Getting other Exposures

```
crop_dataset_infos = client.list_dataset_infos(data_type='crop_production')

client.get_property_values(crop_dataset_infos)
```

```
{'crop': ['whe', 'soy', 'ric', 'mai'],
 'irrigation_status': ['noirr', 'firr'],
 'unit': ['USD', 'Tonnes'],
 'spatial_coverage': ['global']}
```

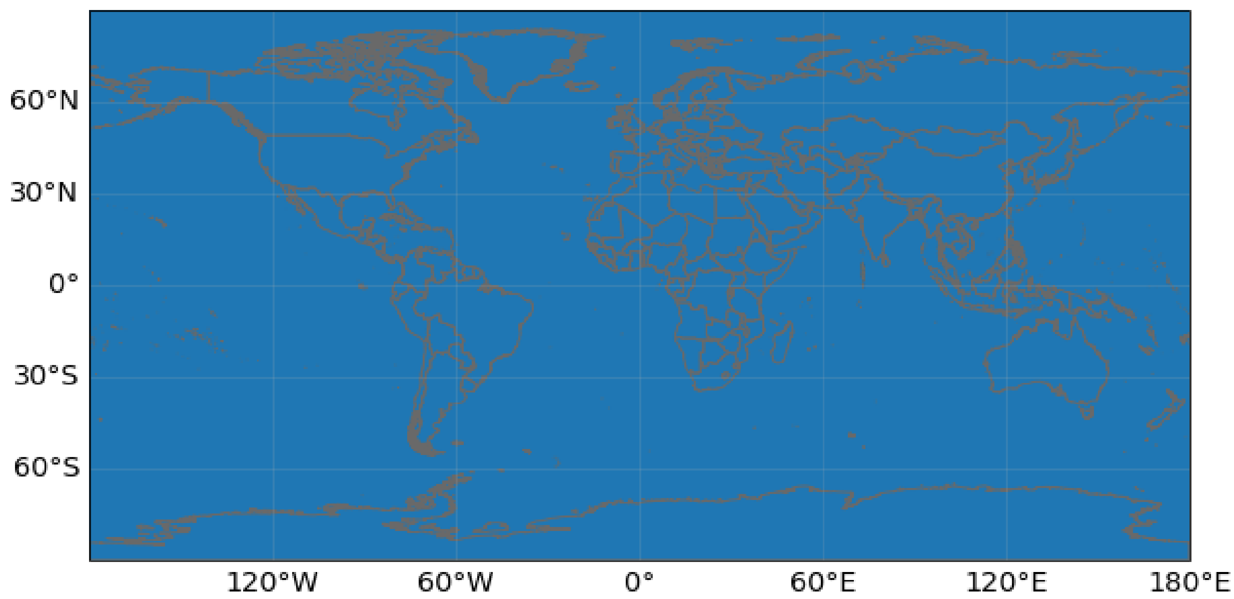
```
rice_exposure = client.get_exposures(exposures_type='crop_production', properties = {
↪'crop':'ric', 'unit': 'USD','irrigation_status': 'noirr'})
```

14.3.3 Getting base centroids to generate new hazard files

```
centroids = client.get_centroids()
centroids.plot()
```

```
https://climada.ethz.ch/data-api/v1/dataset        data_
↪type=centroids        extent=(-180, 180, -90,␣
↪90)        limit=100000        name=None        res_arcsec_land=150        res_
↪arcsec_ocean=1800        status=active        version=None
2022-07-01 15:59:42,013 - climada.hazard.centroids.centr - INFO - Reading /Users/
↪szelie/climada/data/centroids/earth_centroids_150asland_1800asoceans_distcoast_
↪regions/v1/earth_centroids_150asland_1800asoceans_distcoast_region.hdf5
2022-07-01 15:59:44,273 - climada.util.plot - WARNING - Error parsing coordinate␣
↪system 'GEOGCRS["WGS 84",ENSEMBLE["World Geodetic System 1984 ensemble",MEMBER[
↪"World Geodetic System 1984 (Transit)"],MEMBER["World Geodetic System 1984 (G730)"],
↪MEMBER["World Geodetic System 1984 (G873)"],MEMBER["World Geodetic System 1984␣
↪(G1150)"],MEMBER["World Geodetic System 1984 (G1674)"],MEMBER["World Geodetic␣
↪System 1984 (G1762)"],MEMBER["World Geodetic System 1984 (G2139)"],ELLIPSOID["WGS 84
↪",6378137,298.257223563,LENGTHUNIT["metre",1]],ENSEMBLEACCURACY[2.0]],PRIMEM[
↪"Greenwich",0,ANGLEUNIT["degree",0.0174532925199433]],CS[ellipsoidal,2],AXIS[
↪"geodetic latitude (Lat)",north,ORDER[1],ANGLEUNIT["degree",0.0174532925199433]],
↪AXIS["geodetic longitude (Lon)",east,ORDER[2],ANGLEUNIT["degree",0.
↪0174532925199433]],USAGE[SCOPE["Horizontal component of 3D system."],AREA["World."],
↪BBOX[-90,-180,90,180]],ID["EPSG",4326]]'. Using projection PlateCarree in plot.
```

```
<GeoAxesSubplot:>
```



For many hazards, limiting the latitude extent to [-60,60] is sufficient and will reduce the computational ressources required

```
centroids_nopoles = client.get_centroids(extent=[-180,180,-60,50])
centroids_nopoles.plot()
```

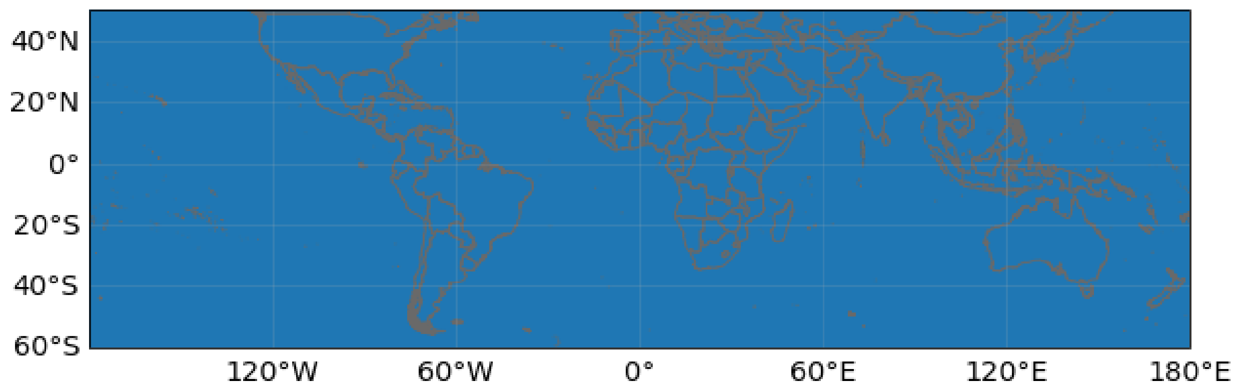
```
https://climada.ethz.ch/data-api/v1/dataset        data_
↪type=centroids        extent=(-180, 180, -90,␣
↪90)        limit=100000        name=None        res_arcsec_land=150        res_
```

(continues on next page)

(continued from previous page)

```
↪arcsec_ocean=1800        status=active        version=None
2022-07-01 15:59:27,602 - climada.hazard.centroids.centr - INFO - Reading /Users/
↪szelie/climada/data/centroids/earth_centroids_150asland_1800asoceans_distcoast_
↪regions/v1/earth_centroids_150asland_1800asoceans_distcoast_region.hdf5
2022-07-01 15:59:29,255 - climada.util.plot - WARNING - Error parsing coordinate␣
↪system 'GEOGCRS["WGS 84",ENSEMBLE["World Geodetic System 1984 ensemble",MEMBER[
↪"World Geodetic System 1984 (Transit)"],MEMBER["World Geodetic System 1984 (G730)"],
↪MEMBER["World Geodetic System 1984 (G873)"],MEMBER["World Geodetic System 1984␣
↪(G1150)"],MEMBER["World Geodetic System 1984 (G1674)"],MEMBER["World Geodetic␣
↪System 1984 (G1762)"],MEMBER["World Geodetic System 1984 (G2139)"],ELLIPSOID["WGS 84
↪",6378137,298.257223563,LENGTHUNIT["metre",1]],ENSEMBLEACCURACY[2.0]],PRIMEM[
↪"Greenwich",0,ANGLEUNIT["degree",0.0174532925199433]],CS[ellipsoidal,2],AXIS[
↪"geodetic latitude (Lat)",north,ORDER[1],ANGLEUNIT["degree",0.0174532925199433]],
↪AXIS["geodetic longitude (Lon)",east,ORDER[2],ANGLEUNIT["degree",0.
↪0174532925199433]],USAGE[SCOPE["Horizontal component of 3D system."],AREA["World."],
↪BBOX[-90,-180,90,180]],ID["EPSG",4326]]'. Using projection PlateCarree in plot.
```

```
<GeoAxesSubplot:>
```



centroids are also available per country:

```
centroids_hti = client.get_centroids(country='HTI')
```

```
https://climada.ethz.ch/data-api/v1/dataset        data_
↪type=centroids        extent=(-180, 180, -90,␣
↪90)        limit=100000        name=None        res_arcsec_land=150        res_
↪arcsec_ocean=1800        status=active        version=None
2022-07-01 16:01:24,328 - climada.hazard.centroids.centr - INFO - Reading /Users/
↪szelie/climada/data/centroids/earth_centroids_150asland_1800asoceans_distcoast_
↪regions/v1/earth_centroids_150asland_1800asoceans_distcoast_region.hdf5
```

14.4 Technical Information

For programmatical access to the CLIMADA data API there is a specific REST call wrapper class: `climada.util.client.Client`.

14.4.1 Server

The CLIMADA data file server is hosted on https://data.iac.ethz.ch that can be accessed via a REST API at https://climada.ethz.ch. For REST API details, see the documentation.

14.4.2 Client

```
Client?
```

```
Init signature: Client()
Docstring:
Python wrapper around REST calls to the CLIMADA data API server.

Init docstring:
Constructor of Client.

Data API host and chunk_size (for download) are configurable values.
Default values are 'climada.ethz.ch' and 8096 respectively.
File:           c:\users\me\polybox\workshop\climada_python\climada\util\api_client.py
Type:           type
Subclasses:
```

```
client = Client()
client.chunk_size
```

```
8192
```

The url to the API server and the chunk size for the file download can be configured in 'climada.conf'. Just replace the corresponding default values:

```
    "data_api": {
        "host": "https://climada.ethz.ch",
        "chunk_size": 8192,
        "cache_db": "{local_data.system}/.downloads.db"
    }
```

The other configuration value affecting the data_api client, `cache_db`, is the path to an SQLite database file, which is keeping track of the files that are successfully downloaded from the api server. Before the Client attempts to download any file from the server, it checks whether the file has been downloaded before and if so, whether the previously downloaded file still looks good (i.e., size and time stamp are as expected). If all of this is the case, the file is simply read from disk without submitting another request.

14.4.3 Metadata

Unique Identifiers

Any dataset can be identified with **data_type**, **name** and **version**. The combination of the three is unique in the API servers' underlying database. However, sometimes the name is already enough for identification. All datasets have a UUID, a universally unique identifier, which is part of their individual url. E.g., the uuid of the dataset https://climada.ethz.ch/rest/dataset/b1c76120-4e60-4d8f-99c0-7e1e7b7860ec is "b1c76120-4e60-4d8f-99c0-7e1e7b7860ec". One can retrieve their meta data by:

```
client.get_dataset_info_by_uuid('b1c76120-4e60-4d8f-99c0-7e1e7b7860ec')
```

```
DatasetInfo(uuid='b1c76120-4e60-4d8f-99c0-7e1e7b7860ec', data_
↪type=DataTypeShortInfo(data_type='litpop', data_type_group='exposures'), name=
↪'LitPop_assets_pc_150arcsec_SGS', version='v1', status='active', properties={'res_
↪arcsec': '150', 'exponents': '(3,0)', 'fin_mode': 'pc', 'spatial_coverage': 'country
↪', 'date_creation': '2021-09-23', 'climada_version': 'v2.2.0', 'country_iso3alpha':
↪'SGS', 'country_name': 'South Georgia and the South Sandwich Islands', 'country_
↪iso3num': '239'}, files=[FileInfo(uuid='b1c76120-4e60-4d8f-99c0-7e1e7b7860ec', url=
↪'https://data.iac.ethz.ch/climada/b1c76120-4e60-4d8f-99c0-7e1e7b7860ec/LitPop_
↪assets_pc_150arcsec_SGS.hdf5', file_name='LitPop_assets_pc_150arcsec_SGS.hdf5',␣
↪file_format='hdf5', file_size=1086488, check_sum=
↪'md5:27bc1846362227350495e3d946dfad5e')], doi=None, description="LitPop asset value␣
↪exposure per country: Gridded physical asset values by country, at a resolution of␣
↪150 arcsec. Values are total produced capital values disaggregated proportionally␣
↪to the cube of nightlight intensity (Lit^3, based on NASA Earth at Night). The␣
↪following values were used as parameters in the LitPop.from_countries() method:{
↪'total_values': 'None', 'admin1_calc': 'False','reference_year': '2018', 'gpw_
↪version': '4.11'}Reference: Eberenz et al., 2020. https://doi.org/10.5194/essd-12-
↪817-2020", license='Attribution 4.0 International (CC BY 4.0)', activation_date=
↪'2021-09-13 09:08:28.358559+00:00', expiration_date=None)
```

or by filtering:

Data Set Status

The datasets of climada.ethz.ch may have the following stati:

- **active**: the default for real life data
- **preliminary**: when the dataset is already uploaded but some information or file is still missing
- **expired**: when a dataset is inactivated again
- **test_dataset**: data sets that are used in unit or integration tests have this status in order to be taken seriously by accident When collecting a list of datasets with `get_datasets`, the default dataset status will be 'active'. With the argument `status=None` this filter can be turned off.

DatasetInfo Objects and DataFrames

As stated above `get_dataset` (or `get_dataset_by_uuid`) return a `DatasetInfo` object and `get_datasets` a list thereof.

```
from climada.util.api_client import DatasetInfo
DatasetInfo?
```

```
Init signature:
DatasetInfo(
    uuid: str,
    data_type: climada.util.api_client.DataTypeShortInfo,
    name: str,
    version: str,
    status: str,
    properties: dict,
    files: list,
    doi: str,
    description: str,
    license: str,
    activation_date: str,
    expiration_date: str,
) -> None
Docstring:      dataset data from CLIMADA data API.
File:           c:\users\me\polybox\workshop\climada_python\climada\util\api_client.py
Type:           type
Subclasses:
```

where files is a list of `FileInfo` objects:

```
from climada.util.api_client import FileInfo
FileInfo?
```

```
Init signature:
FileInfo(
    uuid: str,
    url: str,
    file_name: str,
    file_format: str,
    file_size: int,
    check_sum: str,
) -> None
Docstring:      file data from CLIMADA data API.
File:           c:\users\me\polybox\workshop\climada_python\climada\util\api_client.py
Type:           type
Subclasses:
```

Convert into DataFrame

There are conveinience functions to easily convert datasets into pandas DataFrames, `get_datasets` and `expand_files`:

```
client.into_datasets_df?
```

```
Signature: client.into_datasets_df(dataset_infos)
Docstring:
Convenience function providing a DataFrame of datasets with properties.

Parameters
----------
dataset_infos : list of DatasetInfo
     as returned by list_dataset_infos

Returns
-------
pandas.DataFrame
    of datasets with properties as found in query by arguments
File:      c:\users\me\polybox\workshop\climada_python\climada\util\api_client.py
Type:      function
```

```
from climada.util.api_client import Client
client = Client()
litpop_datasets = client.list_dataset_infos(data_type='litpop', properties={'country_
↪name': 'South Georgia and the South Sandwich Islands'})
litpop_df = client.into_datasets_df(litpop_datasets)
litpop_df
```

```
  data_type data_type_group                                  uuid  \
0    litpop       exposures  b1c76120-4e60-4d8f-99c0-7e1e7b7860ec
1    litpop       exposures  3d516897-5f87-46e6-b673-9e6c00d110ec
2    litpop       exposures  a6864a65-36a2-4701-91bc-81b1355103b5

                             name version  status   doi  \
0  LitPop_assets_pc_150arcsec_SGS      v1  active  None
1        LitPop_pop_150arcsec_SGS      v1  active  None
2            LitPop_150arcsec_SGS      v1  active  None

                                         description  \
0  LitPop asset value exposure per country: Gridd...
1  LitPop population exposure per country: Gridde...
2  LitPop asset value exposure per country: Gridd...

                                     license  \
0  Attribution 4.0 International (CC BY 4.0)
1  Attribution 4.0 International (CC BY 4.0)
2  Attribution 4.0 International (CC BY 4.0)

                     activation_date expiration_date res_arcsec exponents  \
0  2021-09-13 09:08:28.358559+00:00            None        150     (3,0)
1  2021-09-13 09:09:10.634374+00:00            None        150     (0,1)
2  2021-09-13 09:09:30.907938+00:00            None        150     (1,1)

  fin_mode spatial_coverage date_creation climada_version country_iso3alpha  \
0       pc          country    2021-09-23          v2.2.0               SGS
```

(continues on next page)

(continued from previous page)

```
1      pop          country    2021-09-23          v2.2.0               SGS
2       pc          country    2021-09-23          v2.2.0               SGS

                                   country_name country_iso3num
0  South Georgia and the South Sandwich Islands             239
1  South Georgia and the South Sandwich Islands             239
2  South Georgia and the South Sandwich Islands             239
```

14.4.4 Download

The wrapper functions get_exposures or get_hazard fetch the information, download the file and opens the file as a climada object. But one can also just download dataset files using the method `download_dataset` which takes a `DatasetInfo` object as argument and downloads all files of the dataset to a directory in the local file system.

```
client.download_dataset?
```

```
Signature:
client.download_dataset(
    dataset,
    target_dir=WindowsPath('C:/Users/me/climada/data'),
    organize_path=True,
)
Docstring:
Download all files from a given dataset to a given directory.

Parameters
----------
dataset : DatasetInfo
    the dataset
target_dir : Path, optional
    target directory for download, by default `climada.util.constants.SYSTEM_DIR`
organize_path: bool, optional
    if set to True the files will end up in subdirectories of target_dir:
    [target_dir]/[data_type_group]/[data_type]/[name]/[version]
    by default True

Returns
-------
download_dir : Path
    the path to the directory containing the downloaded files,
    will be created if organize_path is True
downloaded_files : list of Path
    the downloaded files themselves

Raises
------
Exception
    when one of the files cannot be downloaded
File:      c:\users\me\polybox\workshop\climada_python\climada\util\api_client.py
Type:      method
```

Cache

The method avoids superfluous downloads by keeping track of all downloads in a sqlite db file. The client will make sure that the same file is never downloaded to the same target twice.

Examples

```
# Let's have a look at an example for downloading a litpop dataset first
ds = litpop_datasets[0]  # litpop_datasets is a list and download_dataset expects a␣
↪single object as argument.
download_dir, ds_files = client.download_dataset(ds)
ds_files[0], ds_files[0].is_file()
```

```
(WindowsPath('C:/Users/me/climada/data/exposures/litpop/LitPop_assets_pc_150arcsec_
↪SGS/v1/LitPop_assets_pc_150arcsec_SGS.hdf5'),
 True)
```

```
# Another example for downloading a hazard (tropical cyclone) dataset
ds_tc = tc_dataset_infos[0]
download_dir, ds_files = client.download_dataset(ds_tc)
ds_files[0], ds_files[0].is_file()
```

```
(PosixPath('/home/yuyue/climada/data/hazard/tropical_cyclone/tropical_cyclone_50synth_
↪tracks_150arcsec_rcp26_BRA_2040/v1/tropical_cyclone_50synth_tracks_150arcsec_rcp26_
↪BRA_2040.hdf5'),
 True)
```

If the dataset contains only one file (which is most commonly the case) this file can also be downloaded and accessed in a single step, using the `get_dataset_file` method:

```
from climada.util.api_client import Client
Client().get_dataset_file(
    data_type='litpop',
    properties={'country_name': 'South Georgia and the South Sandwich Islands', 'fin_
↪mode': 'pop'})
```

```
WindowsPath('C:/Users/me/climada/data/exposures/litpop/LitPop_pop_150arcsec_SGS/v1/
↪LitPop_pop_150arcsec_SGS.hdf5')
```

Local File Cache

By default, the API Client downloads files into the `~/climada/data` directory.

In the course of time obsolete files may be accumulated within this directory, because there is a newer version of these files available from the CLIMADA data API, or because the according dataset got expired altogether.
To prevent file rot and free disk space, it's possible to remove all outdated files at once, by simply calling `Client().purge_cache()`. This will remove all files that were ever downloaded with the `api_client.Client` and for which a newer version exists, even when the newer version has not been downloaded yet.

14.4.5 Offline Mode

The API Client is silently used in many methods and functions of CLIMADA, including the installation test that is run to see whether the CLIMADA installation was successful. Most methods of the client send GET requests to the API server assuming the latter is accessible through a working internet connection. If this is not the case, the functionality of CLIMADA is severely limited if not altogether lost. Often this is an unnecessary restriction, e.g., when a user wants to access a file through the API Client that is already downloaded and available in the local filesystem.

In such cases the API Client runs in *offline mode*. In this mode the client falls back to previous results for the same call in case there is no internet connection or the server is not accessible.

To turn this feature off and make sure that all results are current and up to date - at the cost of failing when there is no internet connection - one has to disable tha *cache*. This can be done programmatically, by initializing the API Client with the optional argument `cache_enabled`:

```
client = Client(cache_enabled=False)
```

Or it can be done through configuration. Edit the `climada.conf` file in the working directory or in ~/climada/ and change the "cache_enabled" value, like this:

```
...
    "data_api": {
        ...
        "cache_enabled": false
    },
...
```

While `cache_enabled` is `true` (default), every result from the server is stored as a json file in ~/climada/data/.apicache/ by a unique name derived from the method and arguments of the call. If the very same call is made again later, at a time where the server is not accessible, the client just comes back to the cached result from the previous call.

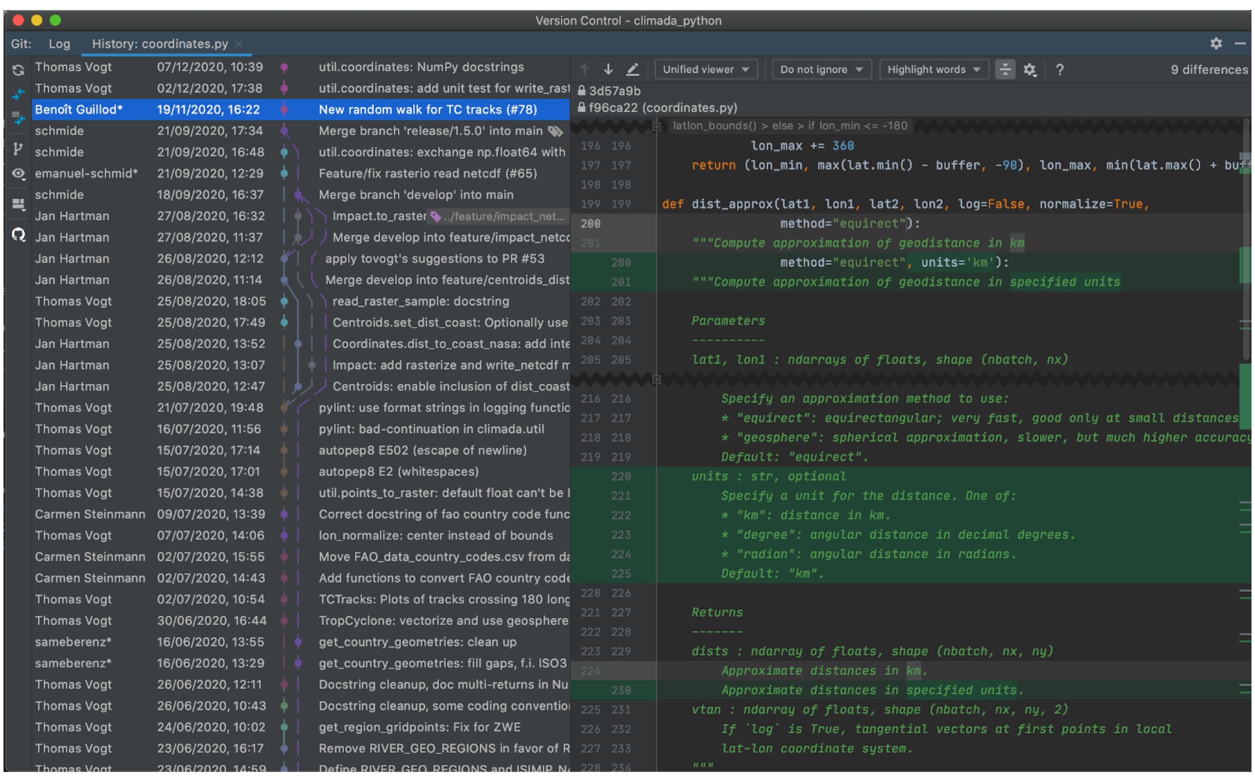
CHAPTER

FIFTEEN

DEVELOPMENT AND GIT AND CLIMADA

15.1 Git and GitHub

- Git's not that scary
  - 95% of your work on Git will be done with the same handful of commands (the other 5% will always be done with careful Googling)
  - Almost everything in Git can be undone by design (but use `rebase`, `--force` and `--hard` with care!)
  - Your favourite IDE (Spyder, PyCharm, …) will have a GUI for working with Git, or you can download a standalone one.
- The Git Book is a great introduction to how Git works and to using it on the command line.
- Consider using a GUI program such as "git desktop" or "Gitkraken" to have a visual git interface, in particular at the beginning. Your python IDE is also likely to have a visual git interface.
- Feel free to ask for help



15.1.1 What we assume you know

We're assuming you're all familiar with the basics of Git.

- What (and why) is version control
- How to clone a repository
- How to make a commit and push it to GitHub
- What a branch is, and how to make one
- How to merge two branches
- The basics of the GitHub website

If you're not feeling great about this, we recommend

- sending me a message so we can arrange an introduction with CLIMADA
- exploring the Git Book

15.1.2 Terms we'll be using today

These are terms that will come up a lot, so let's make sure we know them

- local versus remote
  - Our **remote** repository is hosted on GitHub. This is the central location where all updates to CLIMADA that we want to share end up. If you're updating CLIMADA for the community, your code will end up here too.
  - Your **local** repository is the copy you have on the machine you're working on, and where you do your work.
  - Git calls the (first, default) remote the `origin`
  - (It's possible to set more than one remote repository, e.g. you might set one up on a network-restricted computing cluster)
- push, pull and pull request
  - You **push** your work when you send it from your local machine to the remote repository
  - You **pull** from the remote repository to update the code on your local machine
  - A **pull request** is a standardised review process on GitHub. Usually it ends with one branch merging into another
- Conflict resolution
  - Sometimes two people have made changes to the same bit of code. Usually this comes up when you're trying to merge branches. The changes have to be manually compared and the code edited to make sure the 'correct' version of the code is kept.

15.2 Gitflow

Gitflow is a particular way of using git to organise projects that have

- multiple developers
- working on different features
- with a release cycle

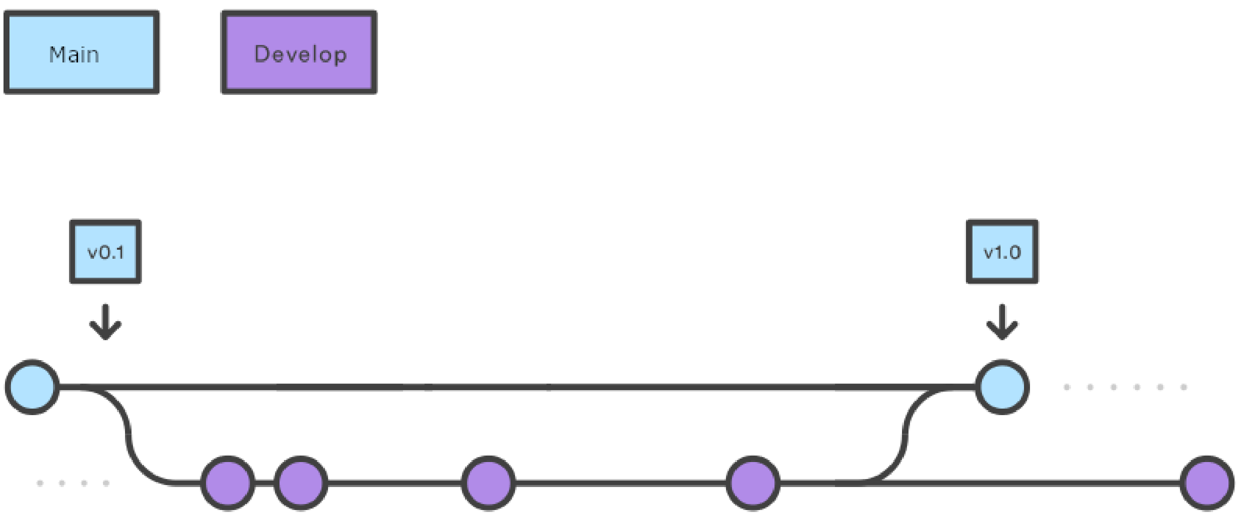
It means that

- there's always a stable version of the code available to the public
- the chances of two developers' code conflicting are reduced
- the process of adding and reviewing features and fixes is more standardised for everyone

Gitflow is a *convention*, so you don't need any additional software.

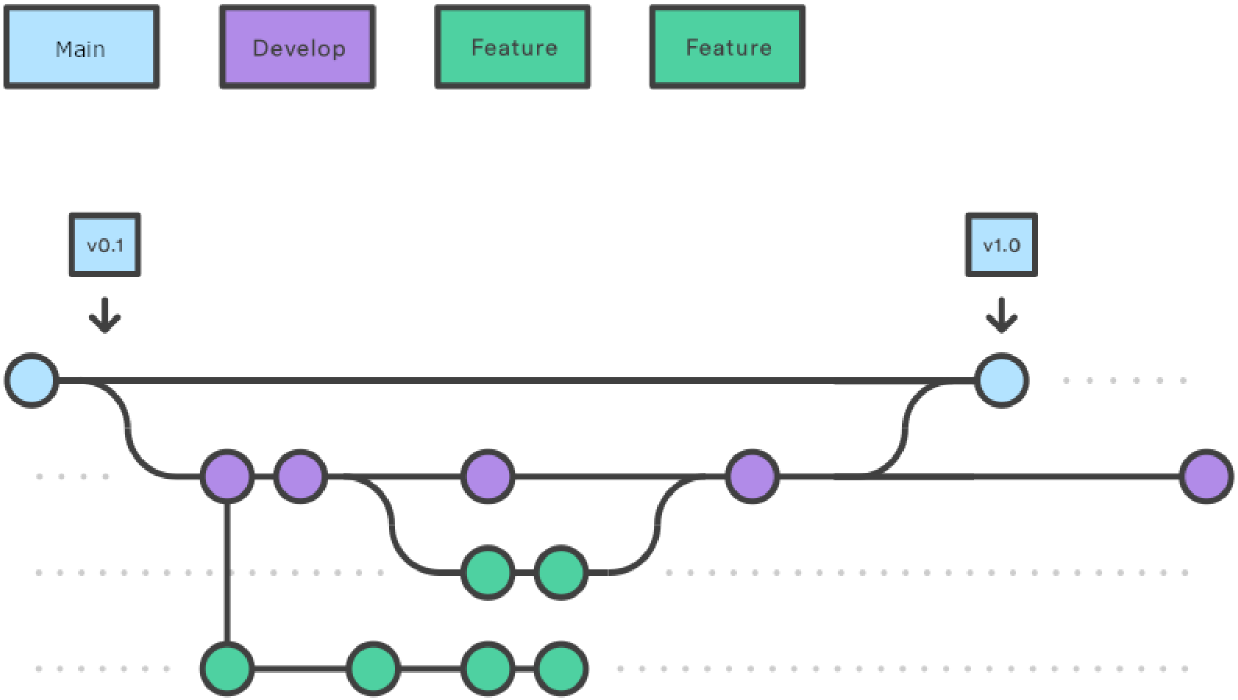
- … but if you want you can get some: a popular extension to the git command line tool allows you to issue more intuitive commands for a Gitflow workflow.
- Mac/Linux users can install git-flow from their package manager, and it's included with Git for Windows

15.2.1 Gitflow works on the `develop` branch instead of `main`



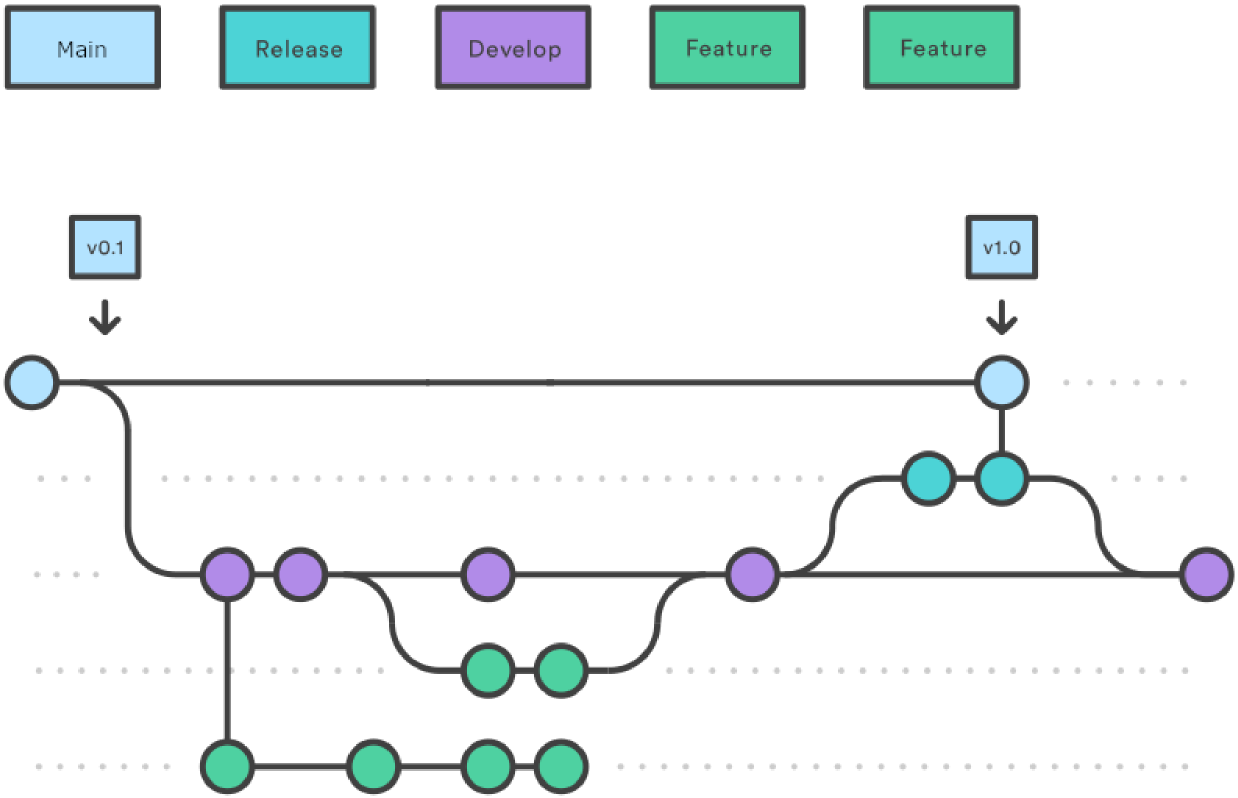
- The critical difference between Gitflow and 'standard' git is that almost all of your work takes place on the `develop` branch, instead of the `main` (formerly `master`) branch.
- The `main` branch is reserved for planned, stable product releases, and it's what the general public download when they install CLIMADA. The developers almost never interact with it.

15.2.2 Gitflow is a feature-based workflow



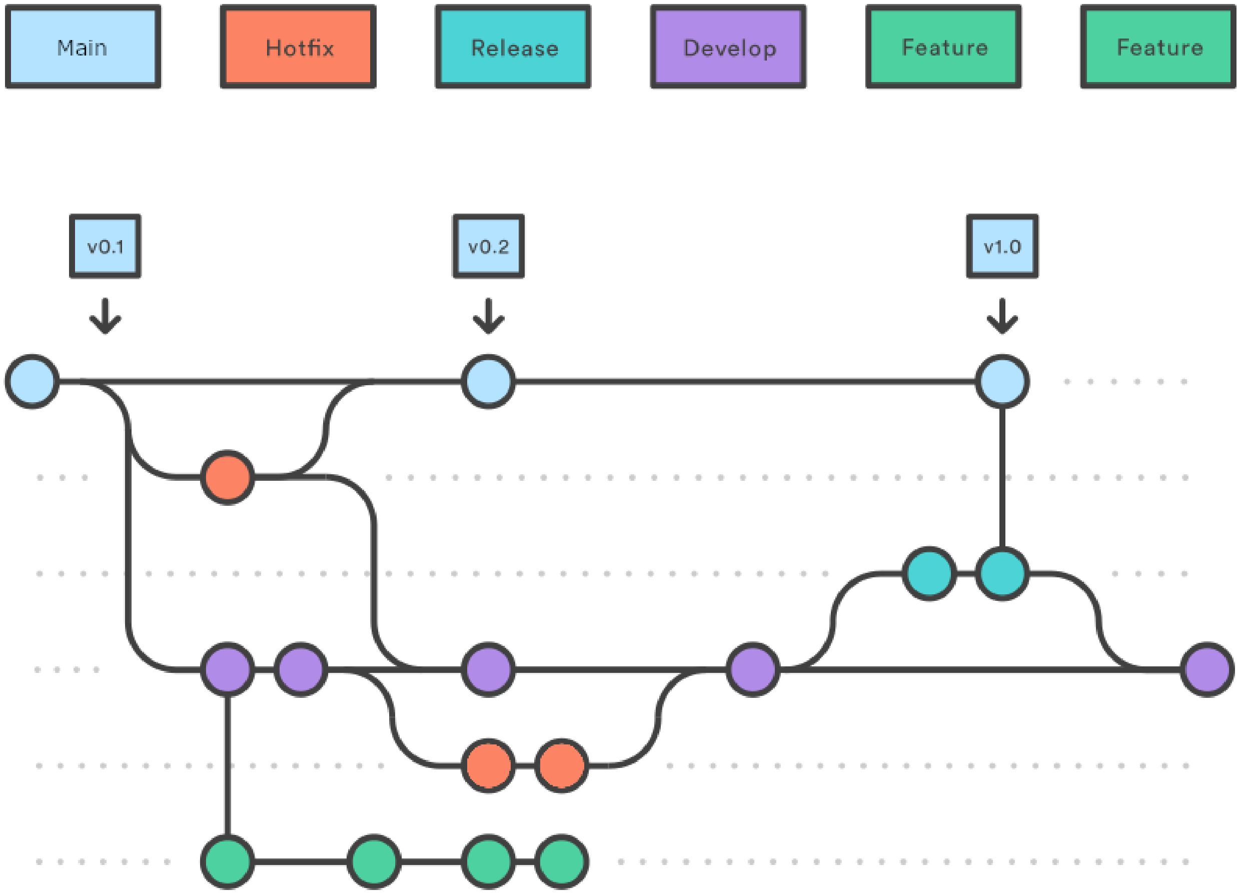
- This is common to many workflows: when you want to add something new to the model you start a new branch, work on it locally, and then merge it back into `develop` **with a pull request** (which we'll cover later).
- By convention we name all CLIMADA feature branches `feature/*` (e.g. `feature/meteorite`).
- Features can be anything, from entire hazard modules to a smarter way to do one line of a calculation. Most of the work you'll do on CLIMADA will be a features of one size or another.
- We'll talk more about developing CLIMADA features later!

15.2.3 Gitflow enables a regular release cycle



- A release is usually more complex than merging `develop` into `main`.
- So for this a `release-*` branch is created from `develop`. We'll all be notified repeatedly when the deadline is to submit (and then to review) pull requests so that you can be included in a release.
- The core developer team (mostly Emanuel) will then make sure tests, bugfixes, documentation and compatibility requirements are met, merging any fixes back into `develop`.
- On release day, the release branch is merged into `main`, the commit is tagged as a release and the release notes are published on the GitHub at https://github.com/CLIMADA-project/climada_python/releases

15.2.4 Everything else is hotfixes



- The other type of branch you'll create is a hotfix.
- Hotfixes are generally small changes to code that do one thing, fixing typos, small bugs, or updating docstrings. They're done in much the same way as features, and are usually merged with a pull request.
- The difference between features and hotfixes is fuzzy and you don't need to worry about getting it right.
- Hotfixes will occasionally be used to fix bugs on the `main` branch, in which case they will merge into both `main` and `develop`.
- Some hotfixes are so simple - e.g. fixing a typo or a docstring - that they don't need a pull request. Use your judgement, but as a rule, if you change what the code does, or how, you should be merging with a pull request.

15.3 Installing CLIMADA for development

See *Installation* for instructions on how to install CLIMADA for developers. You might need to install additional environments contained in `climada_python/requirements` when using specific functionalities. Also see *Apps for working with CLIMADA* for an overview of which tools are useful for CLIMADA developers.

15.4 Does it belong in CLIMADA?

When developing for CLIMADA, it is important to distinguish between core content and particular applications. Core content is meant to be included into the climada_python repository and will be subject to a code review. Any new addition should first be discussed with one of the repository admins. The purpose of this discussion is to see

- How does the planned module fit into CLIMADA?
- What is an optimal architecture for the new module?
- What parts might already exist in other parts of the code?

Applications made with CLIMADA, such as an ECA study can be stored in the paper repository once they have been published. For other types of work, consider making a separate repository that imports CLIMADA as an external package.

15.5 Features and branches

15.5.1 Planning a new feature

Here we're talking about large features such as new modules, new data sources, or big methodological changes. Any extension to CLIMADA that might affect other developers' work, modify the CLIMADA core, or need a big code review.

Smaller feature branches don't need such formalities. Use your judgment, and if in doubt, let people know.

15.5.2 Talk to the group

- Before starting coding a module, do not forget to coordinate with one of the repo admins (Emanuel, Chahan or Lukas)
- This is the chance to work out the Big Picture stuff that is better when it's planned with the group - possible intersections with other projects, possible conflicts, changes to the CLIMADA core, additional dependencies
- Also talk with others from the core development team (see the GitHub wiki).
- Bring it to a developers meeting - people may be able to help/advise and are always interested in hearing about new projects. You can also find reviewers!
- Also, keep talking! Your plans *will* change :)

15.5.3 Planning the work

- Does the project go in its own repository and import CLIMADA, or does it extend the main CLIMADA repository?
  - The way this is done is slowly changing, so definitely discuss it with the group.
  - Chahan will discuss this later!
- Find a few people who will help to review your code.
  - Ask in a developers' meeting, on Slack (for WCR developers) or message people on the development team (see the GitHub wiki).
  - Let them know roughly how much code will be in the reviews, and when you'll be creating pull requests.
- How can the work split into manageable chunks?

  - A series of smaller pull requests is far more manageable than one big one (and takes off some of the pre-release pressure)
  - Reviewing and spotting issues/improvements/generalisations early is always a good thing.
  - It encourages modularisation of the code: smaller self-contained updates, with documentation and tests.
- Will there be any changes to the CLIMADA core?
  - These should be planned carefully
- Will you need any new dependencies? Are you sure?
  - Chahan will discuss this later!

15.5.4 Working on feature branches

When developing a big new feature, consider creating a feature branch and merging smaller branches into that feature branch with pull requests, keeping the whole process separate from `develop` until it's completed. This makes step-by-step code review nice and easy, and makes the final merge more easily tracked in the history.

e.g. developing the big `feature/meteorite` module you might write `feature/meteorite-hazard` and merge it in, then `feature/meteorite-impact`, then `feature/meteorite-stochastic-events` etc… before finally merging `feature/meteorite` into `develop`. Each of these could be a reviewable pull request.

15.5.5 Make a new branch

For new features in Git flow:

```
git flow feature start feature_name
```

Which is equivalent to (in vanilla git):

```
git checkout -b feature/feature_name
```

Or work on an existing branch:

```
git checkout -b branch_name
```

15.5.6 Follow the python do's and don't and performance guides. Write small readable methods, classes and functions.

get the latest data from the remote repository and update your branch

```
git pull
```

see your locally modified files

```
git status
```

add changes you want to include in the commit

```
git add climada/modified_file.py climada/test/test_modified_file.py
```

commit the changes

```
git commit -m "new functionality of .. implemented"
```

15.5.7 Make unit and integration tests on your code, preferably during development

see *Guide on unit and integration tests*

15.6 Pull requests

We want every line of code that goes into the CLIMADA repository to be reviewed!

Code review:

- catches bugs (there are *always* bugs)
- lets you draw on the experience of the rest of the team
- makes sure that more than one person knows how your code works
- helps to unify and standardise CLIMADA's code, so new users find it easier to read and navigate
- creates an archived description and discussion of the changes you've made

15.6.1 When to make a pull request

- When you've finished writing a big new class or method (and its tests)
- When you've fixed a bug or made an improvement you want to merge
- When you want to merge a change of code into `develop` or `main`
- When you want to *discuss* a bit of code you've been working on - pull requests aren't only for merging branches

Not all pull requests have to be into `develop` - you can make a pull request into any active branch that suits you.

Pull requests need to be made latest two weeks before a release, see releases.

15.6.2 Step by step pull request!

Let's suppose you've developed a cool new module on the `feature/meteorite` branch and you're ready to merge it into `develop`.

15.6.3 Checklist before you start

- Documentation
- Tests
- Tutorial (if a complete new feature)
- Updated dependencies (if need be)
- Added your name to the AUTHORS file

- Added an entry to the CHANGELOG.md file. See https://keepachangelog.com for information on how this shoud look like.
- (Advanced, optional) interactively rebase/squash recent commits that *aren't yet on GitHub*.

15.6.4 Steps

1) Make sure the develop branch is up to date on your own machine

```
git checkout develop
git pull
```

2) Merge develop into your feature branch and resolve any conflicts

```
git checkout feature/meteorite
git merge develop
```

In the case of more complex conflicts, you may want to speak with others who worked on the same code. Your IDE should have a tool for conflict resolution.

3) Check all the tests pass locally

```
make unit_test
make integ_test
```

4) Perform a static code analysis using pylint with CLIMADA's configuration .pylintrc (in the climada root directory). Jenkins executes it after every push.
   To do it locally, your IDE probably provides a tool, or you can run make lint and see the output in pylint.log.

5) Push to GitHub. If you're pushing this branch for the first time, use

```
git push -u origin feature/meteorite
```

   and if you're updating a branch that's already on GitHub:

```
git push
```

6) Check all the tests pass on the WCR Jenkins server (https://ied-wcr-jenkins.ethz.ch). See Emanuel's presentation for how to do this! You should regularly be pushing your code and checking this!

7) Create the pull request!
   - On the CLIMADA GitHub page, navigate to your feature branch (there's a drop-down menu above the file structure, pointing by default to main).
   - Above the file structure is a branch summary and an icon to the right labelled "Pull request".
   - Choose which branch you want to merge with. This will usually be develop, but may be another feature branch for more complex feature development.
   - Give your pull request an informative title (like a commit message).
   - Write a description of the pull request. This can usually be adapted from your branch's commit messages (you wrote informative commit messages, didn't you?), and should give a high-level summary of the changes, specific points you want the reviewers' input on, and explanations for decisions you've made. The code documentation (and any references) should cover the more detailed stuff.

- Assign reviewers in the page's right hand sidebar. Tag anyone who might be interested in reading the code. You should already have found one or two people who are happy to read the whole request and sign it off (they could also be added to 'Assignees').
- Create the pull request.
- Contact the reviewers to let them know the request is live. GitHub's settings mean that they may not be alerted automatically. Maybe also let people know on the WCR Slack!

8) Talk with your reviewers
   - Use the comment/chat functionality within GitHub's pull requests - it's useful to have an archive of discussions and the decisions made.
   - Take comments and suggestions on board, but you don't need to agree with everything and you don't need to implement everything.
   - If you feel someone is asking for too many changes, prioritise, especially if you don't have time for complex rewrites.
   - If the suggested changes and or features don't block functionality and you don't have time to fix them, they can be moved to Issues.
   - Chase people up if they're slow. People are slow.
9) Once you implement the requested changes, respond to the comments with the corresponding commit implementing each requested change.
10) If the review takes a while, remember to merge `develop` back into the feature branch every now and again (and check the tests are still passing on Jenkins).
    Anything pushed to the branch is added to the pull request.
11) Once everyone reviewing has said they're satisfied with the code you can merge the pull request using the GitHub interface.
    Delete the branch once it's merged, there's no reason to keep it. (Also try not to re-use that branch name later.)
12) Update the `develop` branch on your local machine.

Also see the ***Reviewer Guide*** and ***Reviewer Checklist***!

15.7 General tips and tricks

15.7.1 Ask for help with Git

- Git isn't intuitive, and rewinding or resetting is always work. If you're not certain what you're doing, or if you think you've messed up, send someone a message.

15.7.2 Don't push or commit to develop or main

- Almost all new additions to CLIMADA should be merged into the `develop` branch with a pull request.
- You won't merge into the `main` branch, except for emergency hotfixes (which should be communicated to the team).
- You won't merge into the `develop` branch without a pull request, except for small documentation updates and typos.
- The above points mean you should never need to push the `main` or `develop` branches.

So if you find yourself on the `main` or `develop` branches typing `git merge ...` or `git push` stop and think again - you should probably be making a pull request.

This can be difficult to undo, so contact someone on the team if you're unsure!

15.7.3 Commit more often than you think, and use informative commit messages

- Committing often makes mistakes less scary to undo

```
git reset --hard HEAD
```

- Detailed commit messages make writing pull requests really easy
- Yes it's boring, but *trust me*, everyone (usually your future self) will love you when they're rooting through the git history to try and understand why something was changed

15.7.4 Commit message syntax guidelines

Basic syntax guidelines taken from here https://chris.beams.io/posts/git-commit/ (on 17.06.2020)

- Limit the subject line to 50 characters
- Capitalize the subject line
- Do not end the subject line with a period
- Use the imperative mood in the subject line (e.g. "Add new tests")
- Wrap the body at 72 characters (most editors will do this automatically)
- Use the body to explain what and why vs. how
- Separate the subject from body with a blank line (This is best done with a GUI. With the command line you have to use text editor, you cannot do it directly with the git command)
- Put the name of the function/class/module/file that was edited
- When fixing an issue, add the reference gh-ISSUENUMBER to the commit message e.g. "fixes gh-40." or "Closes gh-40." For more infos see here https://docs.github.com/en/enterprise/2.16/user/github/managing-your-work-on-github/closing-issues-using-keywords#about-issue-references.

15.7.5 What not to commit

There are a lot of things that don't belong in the Git repository:

- Don't commit data, except for config files and very small files for tests.
- Don't commit anything containing passwords or authentication credentials or tokens. (These are annoying to remove from the Git history.) Contact the team if you need to manage authorisations within the code.
- Don't commit anything that can be created by the CLIMADA code itself

If files like this are going to be present for other users as well, add them to the repository's `.gitignore`.

Jupyter Notebook metadata

Git compares file versions by text tokens. Jupyter Notebooks typically contain a lot of metadata, along with binary data like image files. Simply re-running a notebook can change this metadata, which will be reported as file changes by Git. This causes excessive Diff reports that cannot be reviewed conveniently.

To avoid committing changes of unrelated metadata, open Jupyter Notebooks in a text editor instead of your browser renderer. When committing changes, make sure that you indeed only commit things you *did* change, and revert any changes to metadata that are not related to your code updates.

Several code editors use plugins to render Jupyter Notebooks. Here we collect the instructions to inspect Jupyter Notebooks as plain text when using them:

- **VSCode**: Open the Jupyter Notebook. Then open the internal command prompt (`Ctrl` + `Shift` + `P` or `Cmd` + `Shift` + `P` on macOS) and type/select 'View: Reopen Editor with Text Editor'

15.7.6 Log ideas and bugs as GitHub Issues

If there's a change you might want to see in the code - something that generalises, something that's not quite right, or a cool new feature - it can be set up as a GitHub Issue. Issues are pages for conversations about changes to the codebase and for logging bugs, and act as a 'backlog' for the CLIMADA project.

For a bug, or a question about functionality, make a minimal working example, state which version of CLIMADA you are using, and post it with the Issue.

15.7.7 How not to mess up the timeline

Git builds the repository through incremental edits. This means it's great at keeping track of its history. But there are a few commands that *edit* this history, and if histories get out of sync on different copies of the repository you're going to have a bad time.

- Don't rebase any commits that already exist remotely!
- Don't `--force` anything that exists remotely unless you know what you're doing!
- Otherwise, you're unlikely to do anything irreversible
- You can do what you like with commits that only exist on your machine.

That said, doing an interactive rebase to tidy up your commit history *before* you push it to GitHub is a nice friendly gesture :)

15.7.8 Do not fast forward merges

(This shouldn't be relevant - all your merges into `develop` should be through pull requests, which doesn't fast forward. But:)

Don't fast forward your merges unless your branch is a single commit. Use `git merge --no-ff ...`

The exceptions is when you're merging `develop` into your feature branch.

15.7.9 Merge the remote develop branch into your feature branch every now and again

- This way you'll find conflicts early

```
git checkout develop
git pull
git checkout feature/myfeature
git merge develop
```

15.7.10 Create frequent pull requests

I said this already:

- It structures your workflow
- It's easier for reviewers
- If you're going to break something for other people you all know sooner
- It saves work for the rest of the team right before a release

15.7.11 Whenever you do something with CLIMADA, make a new local branch

You never know when a quick experiment will become something you want to save for later.

15.7.12 But do not do everything in the CLIMADA repository

- If you're running CLIMADA rather than developing it, create a new folder, initialise a new repository with `git init` and store your scripts and data there
- If you're writing an extension to CLIMADA that doesn't change the model core, create a new folder, initialise a new repository with `git init` and import CLIMADA. You can always add it to the model later if you need to.

15.7.13 Questions

https://xkcd.com/1597/

CHAPTER

SIXTEEN

CLIMADA TUTORIAL TEMPLATE

16.1 Content

16.2 Why tutorials

Main goal:

The main goal of the tutorials is it to give a complete overview on:

- essential CLIMADA components
- introduce newly developed modules and features

More specifically, tutorials should introduce CLIMADA users to the core functionalities and modules and guide users in their application. Hence, each new module created needs to be accompanied with a tutorial. The following sections give an overview of the basic structure desired for CLIMADA tutorials.

Important:

A tutorial needs to be included with the final pull request for every new feature.

16.3 Basic structure

Every tutorial should cover the following main points. Additional features characteristic to the modules presented can and should be added as seen fit.

16.3.1 Introduction

- What is the feature presented?
  Briefly describe the feature and introduce how it's presented in the CLIMADA framework.
- What is its data structure?
  Present and overview (in the form of a table for example) of where the feature is built into CLIMADA. What class does it belong to, what are the variables of the feature, what is their data structure.
- Table of content:
  How is this tutorial structured?

16.3.2 Illustration of feature functionality and application

Walk users through the core functions of the module and illustrate how the feature can be used. This obviously is dependent on the feature itself. A few core points should be considered when creating the tutorial:

- **SIZE MATTERS!**
  - each notebook as a total should not exceed the critical (yet vague) size of "a couple MB"
  - keep the size of data you use as examples in the tutorial in mind
  - we aim for computational efficiency
  - a lean, well-organized, concise notebook is more informative than a long, messy all-encompassing one.
- follow the general CLIMADA naming convention for the notebook. For example: "climada_hazard_TropCyclone.ipynb"

16.4 Good examples

The following examples can be used as templates and inspiration for your tutorial:

- Exposure tutorial
- Hazard tutorial

16.5 Use only Markdown for headers and table of content

To create headers or a table of content with links, avoid using *hmtl* and prefer instead purely *Markdown* syntax. Follow *Markdown* conventions in the Markdown Guide and the following key points presented in the section below to know how to use correct *Markdown* syntax which is consistent with the rest of CLIMADA documentation. If in doubt, check existing tutorials to see how it is done.

16.5.1 Headers

To structure your tutorial, use headers of different levels to create sections and subsections.

To create an header, write the symbol (#) before your header name

'#' : create a header of level 1

'##' : create a header of level 2

'###' : create a header of level 3

'####' : create a header of level 4

The title of the tutorial should be of level 1 (#), should have its own cell, and should be the first cell of the notebook.

16.6 Table of content

The second cell of the notebook should be the table of content and should have a header name *'Content'* and a level of 2 (##).

To create the table of content, avoid using numbers to list the different sections, prefer instead simple dots by using **'-'** :

-
-

Instead of numbers :

1.
2.

Additionally, the table of content should only contain headers of level 2 (##).

To create a link from the table of content to a certain section of the tutorial, write [Name of your choice] followed by (#Exactly-the-header-name-you-want-to-direct-the-user-to).

Remember to fill white spaces with '-' and that links to headers are case sensitive! (#This-paRt-Is-CaSe-sEnSitIve).

See the syntax of the example below to create a table of content.

16.6.1 Input:

```
## Content

- [My first header](#My-first-header)
- [My second header](#My-second-header)
- [My third header](#My-third-header)
```

16.6.2 Output:

16.7 Content

- *My first header*
- *My second header*
- *My third header*

CHAPTER

SEVENTEEN

CONSTANTS AND CONFIGURATION

17.1 Constants

Constants are values that, once initialized, are never changed during the runtime of a program. In Python constants are assigned to variables with capital letters by convention, and vice versa, variables with capital letters are supposed to be constants.

In principle there are about four ways to define a constant's value:

- *hard coding*: the value is defined in the python code directly
- *argument*: the value is taken from an execution argument
- *context*: the value is derived from the environmental context of the execution, e.g., the current working directory or the date-time of execution start.
- *configuration*: read from a file or database

In CLIMADA, we only use *hard coding* and *configuration* to assign values to constants.

17.1.1 Hard Coded

Hard coding constants is the preferred way to deal with strings that are used to identify objects or files.

```
# suboptimal
my_dict = {'x': 4}
if my_dict['x'] > 3:
    msg = 'well, arh, ...'
msg
```

```
'well, arh, ...'
```

```
# good
X = 'x'
my_dict = {X: 4}
if my_dict[X] > 3:
    msg = 'yeah!'
msg
```

```
'yeah!'
```

```
# possibly overdoing it
X = 'x'
Y = "this doesn't mean that every string must be a constant"
my_dict = {X: 4}
if my_dict[X] > 3:
    msg = Y
msg
```

```
"this doesn't mean that every string must be a constant"
```

```
import pandas as pd
X = 'x'
df = pd.DataFrame({'x':[1,2,3], 'y':[4,5,6]})
try:
    df.X
except:
    from sys import stderr; stderr.write("this does not work\n")
df[X] # this does work but it's less pretty
df.x
```

```
this does not work
```

```
0    1
1    2
2    3
Name: x, dtype: int64
```

17.1.2 Configurable

When it comes to absolute paths, it is urgently suggested to not use hard coded constant values, for obvious reasons. But also relative paths can cause problems. In particular, they may point to a location where the user has not sufficient access permissions. In order to avoid these problems, *all* paths constants in CLIMADA are supposed to be defined through configuration.
→ paths must be configurable

The same applies to urls to external resources, databases or websites. Since they may change at any time, their addresses are supposed to be defined through configuration. Like this it will be possible to access them without the need of tampering with the source code or waiting for a new release.
→ urls must be configurable

Another category of constants that should go into the configuration file are system specifications, such as number of CPU's available for CLIMADA or memory settings.
→ OS settings must be configurable

17.1.3 Where to put constants?

As a general rule, constants are defined in the module where they intrinsically belong to. If they belong equally to different modules though or they are meant to be used globally, there is the module `climada.util.constants` which is compiling constants CLIMADA-wide.

17.2 Configuration

17.2.1 Configuration files

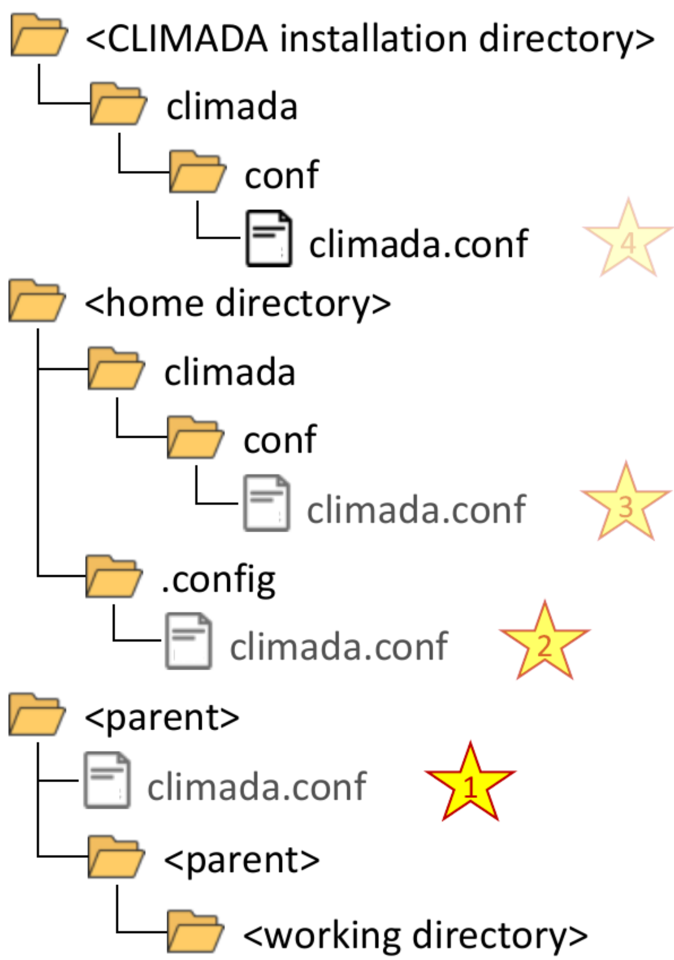
The proper place to define constants that a user may want (or need) to change without changing the CLIMADA installation are the configuration files.
These are files in *json* format with the name `climada.conf`. There is a default config file that comes with the installation of CLIMADA. But it's possible to have several of them. In this case they are complementing one another.

CLIMADA looks for configuration files upon `import climada`. There are four locations to look for configuration files:

- `climada/conf`, the installation directory
- `~/climada/conf`, the user's default climada directory
- `~/.config`, the user's configuration directory,
- `.`, the current working directory

At each location, the path is followed upwards until a file called `climada.conf` is found or the root of the path is reached. Hence, if e.g., `~/climada/climada.conf` is missing but `~/climada.conf` is present, the latter would be read.

When two config files are defining the same value, the priorities are:
`[..]/./climada.conf > ~/.config/climada.conf > ~/climada/conf/climada.conf > installation_dir/climada/conf/climada.conf`



Format

A configuration file is a JSON file, with the additional restriction, that all keys must be strings without a '.' (dot) character .

The JSON format looks a lot like a Python `dict`. But note, that all strings must be surrounded by double quotes and trailing commas are not allowed anywhere.

For configuration values that belong to a particular module it is suggested to reflect the code repositories file structure in the json object. For example, if a configuration for `my_config_value` that belongs to the module `climada.util.dates_times` is wanted, it would be defined as

```
{
  "util": {
    "dates_times": {
      "my_config_value": 42
    }
  }
}
```

Referenced Configuration Values

Configuration string values can be referenced from other configuration values. E.g.

```
{
  "a": "x",
  "b": "{a}y"
}
```

In this example "b" is eventually resolved to "xy".

17.2.2 Accessing configuration values

Configuration values can be accessed through the (constant) `CONFIG` from the `climada` module:

```
from climada import CONFIG
```

```
CONFIG.hazard
```

```
{trop_cyclone: {random_seed: 54}, storm_europe: {forecast_dir: ./results/forecast/
↪hazards}, test_data: .../climada/hazard/test/data}
```

Data Types

The configuration itself and its attributes have the data type `climada.util.config.Config`

```
CONFIG.__class__, CONFIG.hazard.trop_cyclone.random_seed.__class__
```

```
(climada.util.config.Config, climada.util.config.Config)
```

The actual configuration values can be accessed as basic types (bool, float, int, str), provided that the definition is according to the respective data type:

```
CONFIG.hazard.trop_cyclone.random_seed.int()
```

```
54
```

```
try:
    CONFIG.hazard.trop_cyclone.random_seed.str()
except Exception as e:
    from sys import stderr; stderr.write(f"cannot convert random_seed to str: {e}\n")
```

```
cannot convert random_seed to str: <class 'int'>, not str
```

However, configuration string values can be converted to `pathlib.Path` objects if they are pointing to a directory.

```
CONFIG.hazard.storm_europe.forecast_dir.dir()
```

Note that converting a configuration string to a `Path` object like this will create the specified directory on the fly, unless `dir` is called with the parameter `create=False`.

17.2.3 Default Configuration

The conifguration file `climada/conf/climada.conf` contains the default configuration.
On the top level it has the following attributes:

- **local_data**: definition of main paths for accessing and storing CLIMADA related data
  - **system**: top directory, where (persistent) climada data is stored
    default: `~/climada/data`
  - **demo**: top directory for data that is downloaded or created in the CLIMADA tutorials
    default: `~/climada/demo/data`
  - **save_dir**: directory where transient (non-persistent) data is stored
    default: `./results`
- **log_level**: minimum log level showed by logging, one of DEBUG, INFO, WARNING, ERROR or CRITICAL.
  default: `INFO`
- **max_matrix_size**: maximum matrix size that can be used, can be decreased in order to avoid memory issues
  default: `100000000` (1e8)
- **exposures**: exposures modules specific configuration
- **hazard**: hazard modules specific configuration

```
CONFIG.__dict__.keys()
```

```
dict_keys(['_root', '_comment', 'local_data', 'engine', 'exposures', 'hazard', 'util',
↪ 'log_level', 'max_matrix_size', 'data_api', 'test_directory', 'test_data', 'disc_
↪rates', 'impact_funcs', 'measures'])
```

17.2.4 Test Configuration

The configuration values for unit and integration tests are not part of the *default configuration*, since they are irrelevant for the regular CLIMADA user and only aimed for developers.
The default test configuration is defined in the `climada.conf` file of the installation directory. This file contains paths to files that are read during tests. If they are part of the GitHub repository, their path i.g. starts with the `climada` folder within the installation directory:

```
{
    "_comment": "this is a climada configuration file meant to supersede the default␣
↪configuration in climada/conf during test",
    "test_directory": "./climada",
    "test_data": "{test_directory}/test/data",
    "disc_rates": {
        "test_data": "{test_directory}/entity/disc_rates/test/data"
    }
}
```

Obviously, the default `test_directory` is given as the relative path to `./climada`. This is fine if (but only if) unit or integration tests are started from the installation directory, which is the case in the automated tests on the CI server. Developers who intend to start a test from another working directory may have to edit this file and replace the relative path with the absolute path to the installation directory:

```
{
    "_comment": "this is a climada configuration file meant to supersede the default␣
```

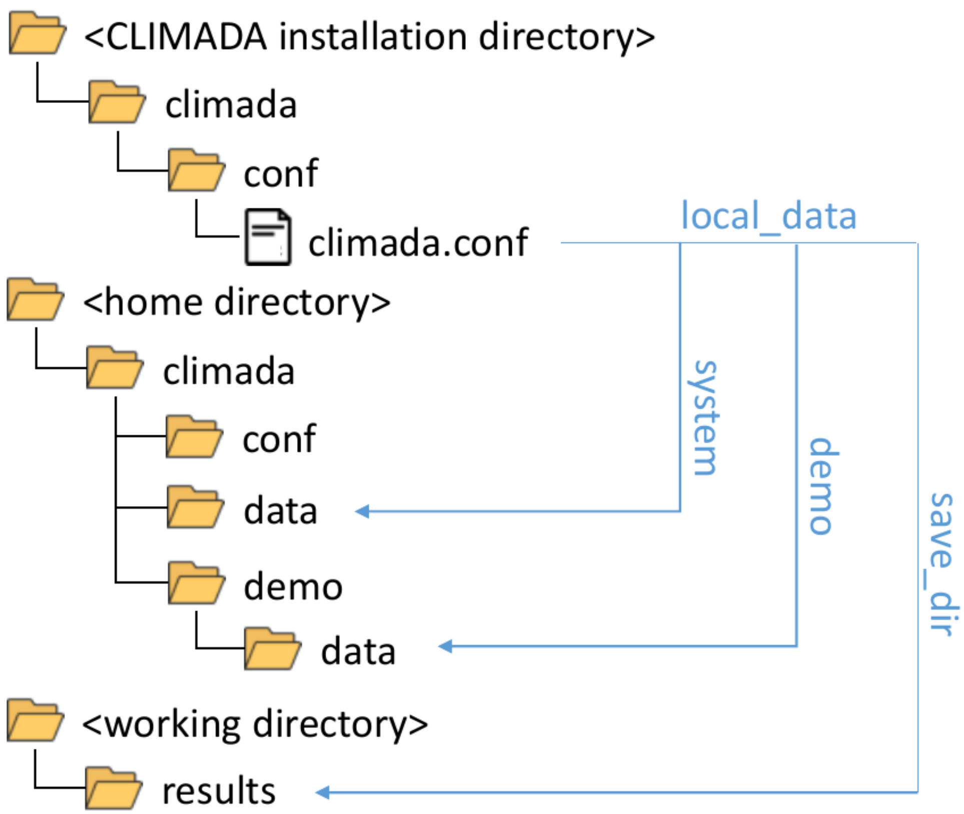
(continues on next page)

(continued from previous page)

```
↪configuration in climada/conf during test",
    "test_directory": "/path/to/installation-dir/climada",
    "test_data": "{test_directory}/test/data",
    "disc_rates": {
        "test_data": "{test_directory}/entity/disc_rates/test/data"
    }
}
```

17.2.5 Data Initialization

When `import climada` is executed in a python script or shell, data files from the installation directory are copied to the location specified in the current configuration.
This happens only when climada is used for the first time with the current configuration. Subsequent execution will only check for presence of files and won't overwrite existing files.



Thus, the home directory will automatically be populated with a climada directory and several files from the repository when climada is used.
To prevent this and keep the home directory clean, create a config file `~/.config/climada.conf` with customized values for `local_data.system` and `local_data.demo`.
As an example, a file with the following content would suppress creation of directories and copying of files during execution of CLIMADA code:

```
{
    "local_data": {
        "system": "/path/to/installation-dir/climada/data/system",
        "demo": "/path/to/installation-dir/climada/data/demo"
    }
}
```

CHAPTER

EIGHTEEN

TESTING

18.1 Notes on Testing

Any programming code that is meant to be used more than once should have a test, i.e., an additional piece of programming code that is able to check whether the original code is doing what it's supposed to do.

Writing tests is work. As a matter of facts, it can be a *lot* of work, depending on the program often more than writing the original code.
Luckily, it essentially follows always the same basic procedure and a there are a lot of tools and frameworks available to facilitate this work.

In CLIMADA we use the Python in-built *test runner* pytest for execution of the tests.

Why do we write test?

- The code is most certainly **buggy** if it's not properly tested.
- Software without tests is **worthless**. It won't be trusted and therefore it won't be used.

When do we write test?

- **Before implementation.** A very good idea. It is called Test Driven Development.
- **During implementation.** Test routines can be used to run code even while it's not fully implemented. This is better than running it interactively, because the full context is set up by the test.
  By command line:
  `python -m unittest climada.x.test_y.TestY.test_z`
  Interactively:
  `climada.x.test_y.TestY().test_z()`
- **Right after implementation.** In case the coverage analysis shows that there are missing tests, see *Test Coverage*.
- **Later, when a bug was encountered.** Whenever a bug gets fixed, also the tests need to be adapted or amended.

18.1.1 Basic Test Procedure

- **Test data setup**
  Creating suitable test data is crucial, but not always trivial. It should be extensive enough to cover all functional requirements and yet as small as possible in order to save resources, both in space and time.
- **Code execution**
  The main goal of a test is to find bugs *before* the user encounters them. Ultimately every single line of the program should be subject to test.
  In order to achieve this, it is necessary to run the code with respect to the whole parameter space. In practice that means that even a simple method may require a lot of test code.

(Bear this in mind when designing methods or functions: the number of required tests increases dramatically with the number of function parameters!)

- **Result validation**
  After the code was executed the *actual* result is compared to the *expected* result. The expected result depends on test data, state and parametrization.
  Therefore result validation can be very extensive. In most cases it won't be practical nor required to validate every single byte. Nevertheless attention should be paid to validate a range of results that is wide enough to discover as many thinkable discrepancies as possible.

18.1.2 Testing types

Despite the common basic procedure there are many different kinds of tests distinguished. (See WikiPedia:Software testing). Very commonly a distinction is made based on levels:

- **Unit Test**: tests only a small part of the code, a single function or method, essentially without interaction between modules
- **Integration Test**: tests whether different methods and modules work well with each other
- **System Test**: tests the whole software at once, using the exposed interface to execute a program

18.1.3 Unit Tests

Unit tests are meant to check the correctness of program units, i.e., single methods or functions, they are supposed to be fast, simple and easy to write.

Developer guidelines:

- **Each module in CLIMADA has a counter part containing unit tests.**
  Naming suggestion: `climada.x.y` → `climada.x.test.test_y`
- **Write a test class for each class of the module, plus a test class for the module itself in case it contains (module) functions.**
  Naming suggestion: `class X` → `class TestX(unittest.TestCase)`, module `climda.x.y` → `class TestY(unittest.TestCase)`
- **Ideally, each method or function should have at least one test method.**
  Naming suggestion: `def xy()` → `def test_xy()`, `def test_xy_suffix1()`, `def test_xy_suffix2()`
  Functions that are created for the sole purpose of structuring the code do not necessarily have their own unit test.
- **Aim at having *very* fast unit tests!**
  There will be hundreds of unit tests and in general they are called in corpore *and expected to finish after a reaonable amount of time.*
  Less than 10 milisecond is good, 2 seconds is the maximum acceptable duration.
- **A unit test shouldn't call more than one climada method or function.**
  The motivation to combine more than one method in a test is usually creation of test data. Try to provide test data by other means. Define them on the spot (within the code of the test module) or create a file in a test data directory that can be read during the test. If this is too tedious, at least move the data acquisition part to the constructor of the test class.
- **Do not use external resources in unit tests.**
  Methods depending on external resources can be skipped from unit tests.

18.1.4 Integration Tests

Integration tests are meant to check the correctness of interaction between units of a module or a package.
As a general rule, more work is required to write integration tests than to write unit tests and they have longer runtime.

Developer guidelines:

- **Write integration tests for all intended use cases**.
- **Do not expect external resources to be immutable**.
  If calling on external resources is part of the workflow to be tested, take into account that they may change over time.
  If the according API has means to indicate the precise version of the requested data, make use of it, otherwise, adapt your expectations and leave room for future changes.
  Example given: your function is ultimately relying on the *current* GDP retrieved from an online data provider, and you test it for Switzerland where it's in about 700 Bio CHF at the moment. Leave room for future development, try to be on a reasonably save side, tolerate a range between 70 Bio CHF and 7000 Bio CHF.
- **Test location**.
  Integration are written in modules `climada.test.test_xy` or in `climada.x.test.test_y`, like the unit tests.
  For the latter it is required that they do not use external resources and that the tests do not have a runtime longer than 2 seconds.

18.1.5 System Tests

System tests are meant to check whether the whole software package is working correctly.

In CLIMADA, the system test that checks the core functionality of the package is executed by calling `make install_test` from the installation directory.

18.1.6 Error Messages

When a test fails, make sure the raised exception contains all information that might be helpful to identify the exact problem.
If the error message is ever going to be read by someone else than you while still developing the test, you best assume it will be someone who is completely naive about CLIMADA.

Writing extensive failure messages will eventually save more time than it takes to write them.

Putting the failure information into logs is neither required nor sufficient: the automated tests are built around error messages, not logs.
Anything written to `stdout` by a test method is useful mainly for the developer of the test.

18.1.7 Test Coverage

Coverage is a measure of how much of your code is actually checked by the tests. One distinguishes between line coverage and branch or conditionals coverage. The line coverage reports the percentage of all lines of code covered by the tests. The branch coverage reports the percentage of all possible branches covered by the tests. Achieving a high branch coverage is much harder than a high line coverage.

In CLIMADA, we aim for a high line coverage (only). Ideally, any new code should have a line coverage of 100%, meaning every line of code is tested. You can inspect the test coverage of your local code by following the instructions for executing tests below.

See the *Continuous Integration Guide* for information on how to inspect coverage of the automated test pipeline.

18.1.8 Test files

For integration tests it can be required to read data from a file, in order to set up a test that aims to check functionality with non-trivial data, beyond the scope of unit tests. Some of thes test files can be found in the `climada/**/test/data` directories or in the `climada/data` directory. As mostly the case with large test data, it is not very well suitable for a Git repository.

The preferable alternative is to post the data to the Climada Data-API with status `test_dataset` and retrieve the files on the fly from there during tests. To do this one can use the convenience method `climada.test.get_test_file`:

```
from climada.test import get_test_file

my_test_file = get_test_file(ds_name='my-test-file', file_format='hdf5')  # returns a␣
↪pathlib.Path object
```

Behind the scenes, `get_test_file` uses the `climada.util.api_client.Client` to identify the appropriate dataset and downloads the respective file to the local dataset cache (`~/climada/data/*`).

18.1.9 Dealing with External Resources

Methods depending on external resources (calls a url or database) are ideally atomic and doing nothing else than providing data. If this is the case they can be skipped in unit tests on safe grounds - provided they are tested at some point in higher level tests.

In CLIMADA there are the utility functions `climada.util.files_handler.download_file` and `climada.util.files_handler.download_ftp`, which are assigned to exactly this task for the case of external data being available as files.

Any other method that is calling such a data providing method can be made compliant to unit test rules by having an option to replace them by another method. Like this one can write a dummy method in the test module that provides data, e.g., from a file or hard coded, which be given as the optional argument.

```
import climada
def x(download_file=climada.util.files_handler.download_file):
    filepath = download_file('http://real_data.ch')
    return Path(filepath).stat().st_size

import unittest
class TestX(unittest.TestCase):
    def download_file_dummy(url):
        return "phony_data.ch"

```

(continues on next page)

(continued from previous page)

```
    def test_x(self):
        self.assertEqual(44, x(download_file=self.download_file_dummy))
```

Developer guideline:

- When introducing a new external resource, add a test method in `test_data_api.py`.

18.1.10 Test Configuration

Use the configuration file `climada.config` in the installation directory to define file paths and external resources used during tests (see the *Constants and Configuration Guide*).

18.2 Testing CLIMADA

Executing the entire test suite requires you to install the additional requirements for testing. See the *installation instructions* for *developer dependencies* for further information.

In general, you execute tests with

```
pytest <path>
```

where you replace `<path>` with a Python file containing tests or an entire directory containing multiple test files. Pytest will walk through all subdirectories of `<path>` and try to discover all tests. For example, to execute *all tests* within the CLIMADA repository, execute

```
pytest climada/
```

from within the `climada_python` directory.

18.2.1 Installation Test

From the installation directory run

```
make install_test
```

It lasts about 45 seconds. If it succeeds, CLIMADA is properly installed and ready to use.

18.2.2 Unit Tests

From the installation directory run

```
make unit_test
```

It lasts about 5 minutes and runs unit tests for all modules.

18.2.3 Integration Tests

From the installation directory run

```
make integ_test
```

It lasts about 15 minutes and runs extensive integration tests, during which also data from external resources is read. An open internet connection is required for a successful test run.

18.2.4 Coverage

Executing `make unit_test` and `make integ_tests` provides local coverage reports as HTML pages at `coverage/index.html`. You can open this file with your browser.

CHAPTER

NINETEEN

CONTINUOUS INTEGRATION AND GITHUB ACTIONS

19.1 Automated Tests

On Jenkins tests are executed and analyzed automatically, in an unbiased environment. The results are stored and can be compared with previous test runs.
Jenkins has a GUI for monitoring individual tests, full test runs and test result trends.
Developers are requested to watch it. At first when they push commits to the code repository, but also later on, when other changes in data or sources may make it necessary to review and refactor code that once passed all tests. The CLIMADA Jenkins server used for continuous integration is at (https://ied-wcr-jenkins.ethz.ch) .

19.1.1 Developer guidelines:

- All tests must pass before submitting a pull request.
- Integration tests don't run on feature branches in Jenkins, therefore developers are requested to run them locally.
- After a pull request was accepted and the changes are merged to the develop branch, integration tests may still fail there and have to be addressed.

19.2 Test Coverage

Jenkins also has an interface for exploring code coverage analysis result.
This shows which part of the code has never been run in any test, by module, by function/method and even by single line of code.

Ultimately every single line of code should be tested.

19.2.1 Jenkins Coverage Reports

To inspect the coverage reports, check out the overview of branch builds on Jenkins. Select the branch or pull request you are interested in. Then, select "Coverage Report" in the menu on the right. Note that this menu entry might not be available if no build of that particular branch/PR succeeded.

You will see a report for every directory and file in CLIMADA. Clicking on a specific file opens a view of the file where the coverage is highlighted.

19.2.2 GitHub Coverage Reports

To inspect the coverage reports for the GitHub Actions (see below), click on the "Checks" tag in a pull request and then on "GitHub CI" on the left. In the summary of all tasks you will find the "Artifacts" with coverage reports provided as ZIP files. You can download these files, unzip them, and open the resulting HTML files in your browser.

19.2.3 Developer guidelines:

- Make sure the coverage of novel code is at 100% before submitting a pull request.

Be aware that having a code coverage alone does not grant that all required tests have been written!
The following artificial example would have a 100% coverage and still obviously misses a test for `y(False)`

```
def x(b:bool):
    if b:
        print('been here')
        return 4
    else:
        print('been there')
        return 0

def y(b:bool):
    print('been everywhere')
    return 1/x(b)


import unittest
class TestXY(unittest.TestCase):
    def test_x(self):
        self.assertEqual(x(True), 4)
        self.assertEqual(x(False), 0)

    def test_y(self):
        self.assertEqual(y(True), 0.25)

unittest.TextTestRunner().run(unittest.TestLoader().loadTestsFromTestCase(TestXY));
```

```
..
```

```
been here
been there
been everywhere
been here
```

```
----------------------------------------------------------------------
Ran 2 tests in 0.003s

OK
```

19.3 Static Code Analysis

At last Jenkins provides an elaborate GUI for pylint findings which is especially useful when working in feature branches.

Observe it!

19.3.1 Developer guidelines:

- *High Priority Warnings* are as severe as test failures and must be addressed at once.
- Do not introduce new *Medium Priority Warnings*.
- Try to avoid introducing *Low Priority Warnings*, in any case their total number should not increase.

19.4 Jenkins Projects Overview

19.4.1 climada_install_env

Branch: **develop**
Runs every day at 1:30AM CET

- creates conda environment from scratch
- runs core functionality system test (`make install_test`)

19.4.2 climada_ci_night

Branch: **develop**
Runs when climada_install_env has finished successfully

- runs all test modules
- runs static code analysis

19.4.3 climada_branches

Branch: **any**
Runs when a commit is pushed to the repository

- runs all test modules *outside of climada.test*
- runs static code analysis

19.4.4 climada_data_api

Branch: **develop**
Runs every day at 0:20AM CET

- tests availability of external data APIs

19.4.5 climada_data_api

Branch: **develop**
No automated running

- tests executability of CLIMADA tutorial notebooks.

19.5 GitHub Actions

CLIMADA has been using a private Jenkins instance for automated testing (Continuous Integration, CI). We recently adopted GitHub Actions for automated unit testing. GitHub Actions is a service provided by GitHub, which lets you configure CI/CD pipelines based on YAML configuration files. GitHub provides servers which ample computational resources to create software environments, install software, test it, and deploy it. See the GitHub Actions Overview for a technical introduction, and the Workflow Syntax for a reference of the pipeline definitions.

The CI results for each pull request can be inspected in the "Checks" tab. For GitHub Actions, users can inspect the logs of every step for every job.

19.5.1 Note

As of CLIMADA v4.0, the default CI technology remains Jenkins. GitHub Actions CI is currently considered experimental for CLIMADA development.

19.5.2 Unit Testing Guideline

This pipeline is defined by the `.github/workflows/ci.yml` file. It contains a single job which will create a CLIMADA environment with Mamba for multiple Python versions, install CLIMADA, run the unit tests, and report the test coverage as well as the simplified test results. The job has a strategy which runs it for multiple times for different Python versions. This way, we make sure that CLIMADA is compatible with all currently supported versions of Python.

The coverage reports in HTML format will be uploaded as job artifacts and can be downloaded as ZIP files. The test results are simple testing summaries that will appear as individual checks/jobs after the respective job completed.

CHAPTER

TWENTY

REVIEWER GUIDELINES

20.1 How to review a pull request

- Be friendly
- Read and follow the ***Reviewer Checklist***
- Decide how much time you can spare and the detail you can work in. Tell the author!
- Use the comment/chat functionality within GitHub's pull requests - it's useful to have an archive of discussions and the decisions made.
- Fix the big things first! If there are more important issues, not every style guide has to be stuck to, not every slight increase in speed needs to be pointed out, and test coverage doesn't have to be 100%.
- Make it clear when a change is optional, or is a matter of opinion

At a minimum

- Make sure unit and integration tests are passing.
- (For complete modules) Run the tutorial on your local machine and check it does what it says it does
- Check everything is fully documented

At least one reviewer needs to

- Review all the changes in the pull request. Read what it's supposed to do, check it does that, and make sure the logic is sound.
- Check that the code follows the CLIMADA style guidelines
- *CLIMADA coding conventions*
- *Python Dos and Don't*
- *Python performance tips and best practice for CLIMADA developers*
- If the code is implementing an algorithm it should be referenced in the documentation. Check it's implemented correctly.
- Try to think of edge cases and ways the code could break. See if there's appropriate error handling in cases where the function might behave unexpectedly.
- (Optional) suggest easy ways to speed up the code, and more elegant ways to achieve the same goal.

There are a few ways to suggest changes

- As questions and comments on the pull request page

- As code suggestions (max a few lines) in the code review tools on GitHub. The author can then approve and commit the changes from GitHub pull request page. This is great for typos and little stylistic changes.
- If you decide to help the author with changes, you can either push them to the same branch, or create a new branch and make a pull request with the changes back into the branch you're reviewing. This lets the author review it and merge.

20.2 Reviewer Checklist

- The code must be readable without extra effort from your part. The code should be easily readable (for infos e.g. here)
- Include references to the used algorithms in the docstring
- If the algorithm is new, please include a description in the docstring, or be sure to include a reference as soon as you publish the work
- Variable names should be chosen to be clear. Avoid `item, element, var, list, data` etc... A good variable name makes it immediately clear what it contains.
- Avoid as much as possible hard-coded indices for list (no `x = l[0], y = l[1]`). Rather, use tuple unpacking (see here). Note that tuple unpacking can also be used to update variables. For example, the Fibonacci sequence next number pair can be written as `n1, n2 = n2, n1+n2`.
- Do not use mutable (lists, dictionaries, ...) as default values for functions and methods. Do not write:

  ```
  def function(default=[]):
  ```

  but use

  ```
  def function(default=None):
      if default is None: default=[]
  ```

- Use pythonic loops, list comprehensions
- Make sure the unit tests are testing all the lines of the code. Do not only check for working cases, but also the most common wrong use cases.
- Check the docstrings (Do they follow the Numpydoc conventions, is everything clearly explained, are the default values given and is it clear why they are set to this value)
- Keep the code simple. Avoid using complex Python functionalities whose use is opaque to non-expert developers unless necessary. For example, the `@staticmethod` decorator should only be used if really necessary. Another example, for counting the dictionary `colors = ['red', 'green', 'red', 'blue', 'green', 'red']`, version:

  ```
  d = {}
  for color in colors:
          d[color] = d.get(color, 0) + 1
  ```

  is perfectly fine, no need to complicate it to a maybe more pythonic version

  ```
  d = collections.defaultdict(int)
  for color in colors:
      d[color] += 1
  ```

- Did the code writer perform a static code analysis? Does the code respect Pep8 (see also the pylint config file)?

- Did the code writer perform a profiling and checked that there are no obviously inefficient (computation time-wise and memory-wise) parts in the code?

CHAPTER

TWENTYONE

CODING IN PYTHON: DOS AND DON'TS

21.1 To Code or Not to Code?

Before you start implementing functions which then go into the climada code base, you have to ask yourself a few questions:

Has something similar already been implemented? This is far from trivial to answer! First, search for functions in the same module where you'd be implementing the new piece of code. Then, search in the `util` folders, there's a lot of functions in some of the scripts! You could also search the index (a list of all functions and global constants) in the climada documentation for key-words that may be indicative of the functionality you're looking for.

Don't expect this process to be fast!

Even if you want to implement *just* a small helper function, which might take 10mins to write, it may take you 30mins to check the existing code base! That's part of the game! Even if you found something, most likely, it's not the *exact* same thing which you had in mind. Then, ask yourself how you can re-use what's there, or whether you can easily add another option to the existing method to also fit your case, and only if it's nearly impossible or highly unreadable to do so, write your own implementation.

Can my code serve others? You probably have a very specific problem in mind. Yet, think about other use-cases, where people may have a similar problem, and try to either directly account for those, or at least make it easy to configure to other cases. Providing keyword options and hard-coding as few things as possible is usually a good thing. For example, if you want to write a daily aggregation function for some time-series, consider that other people might find it useful to have a general function that can also aggregate by week, month or year.

Can I get started? Before you finally start coding, be sure about placing them in a sensible location. Functions in non-util modules are actually specific for that module (e.g. a file-reader function is probably not river-flood specific, so put it into the `util` section, not the `RiverFlood` module, even if that's what you're currently working on)! If unsure, talk with other people about where your code should go.

If you're implementing more than just a function or two, or even an entirely new module, the planning process should be talked over with someone doing climada-administration.

21.2 Clean Code

A few basic principles:

- Follow the PEP 8 Style Guide. It contains, among others, recommendations on:
  - code layout
  - basic naming conventions
  - programming recommendations

  - commenting (in detail described in Chapter 4)
  - varia
- Perform a static code analysis - or: PyLint is your friend
- Follow the best practices of *Correctness - Tightness - Readability*
- Adhere to principles of pythonic coding (idiomatic coding, the “python way”)

The Zen of Python

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do
it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

21.2.1 PEP 8 Quickie: Code Layout

- *Indentation*: 4 spaces per level. For continuation lines, decide between vertical alignment & hanging indentation as shown here:

```
# Vertically aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Hanging indentation (4 additonal spaces)
def very_very_long_function_name(
        var_one, var_two, var_three,
```

(continues on next page)

(continued from previous page)

```
        var_four):
    print(var_one)
```

- *Line limit*: maximum of 79 characters (docstrings & comments 72).
- *Blank lines*:
  - **Two**: Surround top-level function and class definitions;
  - **One:** Surround Method definitions inside a class
  - **Several:** may be used (sparingly) to separate groups of related functions
  - **None:** Blank lines may be omitted between a bunch of related one-liners (e.g. a set of dummy implementations).
- *Whitespaces*:
  - **None** immediately inside parentheses, brackets or braces; after trailing commas; for keyword assignments in functions.
  - **Do** for assignments (`i = i + 1`), around comparisons (>=, ==, etc.), around booleans (`and`, `or`, `not`)
  - the following 3 examples are correct:

```
spam(ham[1], {eggs: 2})
if x == 4: print x, y; x, y = y, x
def complex(real, imag=0.0):
```

- There's more in the PEP 8 guide!

21.2.2 PEP 8 Quickie: Basic Naming Conventions

A short typology: b (single lowercase letter); B (single uppercase letter); lowercase; lower_case_with_underscores; UPPERCASE; UPPER_CASE_WITH_UNDERSCORES; CapitalizedWords (or CapWords, or CamelCase); mixedCase; Capitalized_Words_With_Underscores (ugly!)

A few basic rules:

- packages and modules: short, all-lowercase names. Underscores can be used in the module name if it improves readability. E.g. `numpy`, `climada`
- classes: use the CapWords convention. E.g. `RiverFlood`
- functions, methods and variables: lowercase, with words separated by underscores as necessary to improve readability. E.g. `from_raster()`, `dst_meta`
- function- and method arguments: Always use `self` for the first argument to instance methods,`cls` for the first argument to class methods.
- constants: all capital letters with underscores, e.g. `DEF_VAR_EXCEL`

Use of underscores

- `_single_leading_underscore`: weak "internal use" indicator. E.g. `from M import *` does not import objects whose names start with an underscore. A side-note to this: Always decide whether a class's methods and instance variables (collectively: "attributes") should be public or non-public. If in doubt, choose non-public; it's easier to make it public later than to make a public attribute non-public. Public attributes are those that you expect unrelated clients of your class to use, with your commitment to avoid backwards incompatible changes. Non-public attributes are those that are not intended to be used by third parties; you make no guarantees that non-public attributes won't change or even be removed. Public attributes should have no leading underscores.

- `single_trailing_underscore_`: used by convention to avoid conflicts with Python keyword, e.g. `tkinter.Toplevel(master, class_='ClassName')`
- `__double_leading_and_trailing_underscore__`: "magic" objects or attributes that live in user-controlled namespaces. E.g. `__init__`, `__import__` or `__file__`. Never invent such names; only use them as documented.

There are many more naming conventions, some a bit messy. Have a look at the PEP8 style guide for more cases.

21.2.3 PEP 8 Quickie: Programming Recommendations

- comparisons to singletons like `None` should always be done with `is` or `is not`, never the equality operators.
- Use `is not` operator rather than `not ... is`.
- Be consistent in return statements. Either all return statements in a function should return an expression, or none of them should. Any return statements where no value is returned should explicitly state this as `return None`.

```
# Correct
def foo(x):
    if x >= 0:
        return math.sqrt(x)
    else:
        return None
# Wrong
def foo(x):
    if x >= 0:
        return math.sqrt(x)
```

- Object type comparisons should always use isinstance() instead of comparing types directly:

```
# Correct:
if isinstance(obj, int):
# Wrong:
if type(obj) is type(1)
```

- Remember: sequences (strings, lists, tuples) are false if empty; this can be used:

```
# Correct:
if not seq:
if seq:
# Wrong:
if len(seq):
if not len(seq)
```

- Don't compare boolean values to True or False using ==:

```
# Correct:
if greeting:
# Wrong:
if greeting == True:
```

- Use ".startswith() and ".endswith() instead of string slicing to check for prefixes or suffixes.

```
# Correct:
if foo.startswith('bar'):
# Wrong:
if foo[:3] == 'bar':
```

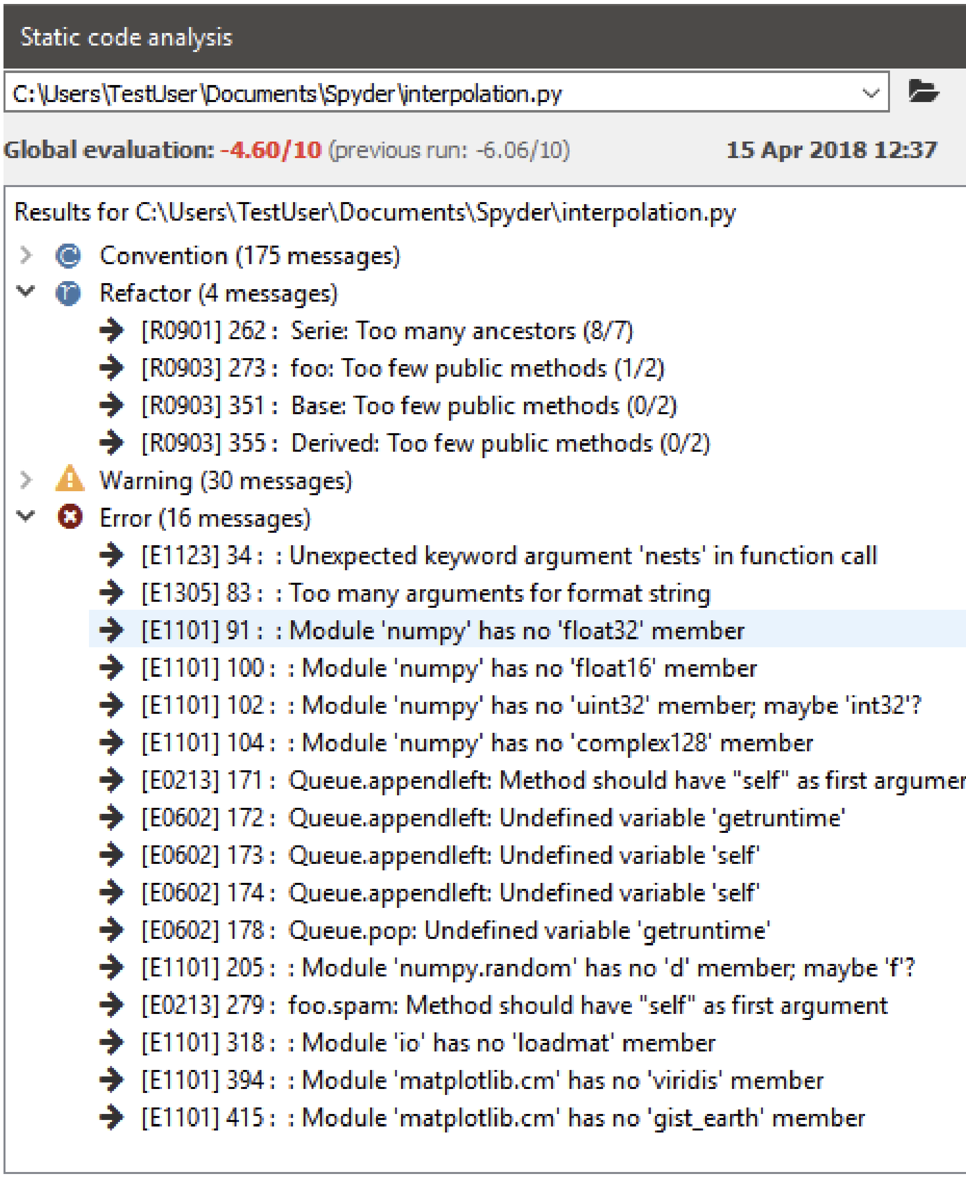
- Context managers exist and can be useful (mainly for opening and closing files

21.2.4 Static Code Analysis and PyLint

Static code analysis detects style issues, bad practices, potential bugs, and other quality problems in your code, all without having to actually execute it. In Spyder, this is powered by the best in class Pylint back-end, which can intelligently detect an enormous and customizable range of problem signatures. It follows the style recommended by PEP 8 and also includes the following features: Checking the length of each line, checking that variable names are well-formed according to the project's coding standard, checking that declared interfaces are truly implemented.

A detailed instruction can be found here.

In brief: In the editor, select the Code Analysis pane (if not visible, go to View -> Panes -> Code Analysis) and the file you want to be analyzed; hit the Analyze button.



The output will look somewhat similar to that:

There are 4 categories in the analysis output:

- *convention*,
- *refactor*,
- *warning*,
- *error*
- a global score regarding code quality.

All messages have a line reference and a short description on the issue. Errors *must* be fixed, as this is a no-go for actually executing the script. Warnings and refactoring messages should be taken seriously; so should be the convention messages, even though some of the naming conventions etc. may not fit the project style. This is configurable.

In general, there should be no errors and warnings left, and the overall code quality should be in the "green" range (somewhere above 5 or so).

There are advanced options to configure the type of warnings and other settings in pylint.

21.2.5 A few more best practices

Correctness

Methods and functions must return correct and verifiable results, not only under the best circumstances but in any possible context. I.e. ideally there should be unit tests exploring the full space of parameters, configuration and data states. This is often clearly a non-achievable goal, but still - we aim at it.

Tightness

- Avoid code redundancy.
- Make the program efficient, use profiling tools for detection of bottlenecks.
- Try to minimize memory consumption.
- Don't introduce new dependencies (library imports) when the desired functionality is already covered by existing dependencies.
- Stick to already supported file types.

Readability

- Write complete Python Docstrings.
- Use meaningful method and parameter names, and always annotate the data types of parameters and return values.
- No context-dependent return types! Also: Avoid `None` as return type, rather raise an Exception instead.
- Be generous with defining Exception classes.
- Comment! Comments are welcome to be redundant. And whenever there is a particular reason for the way something is done, comment on it! See below for more detail.
- For functions which implement mathematical/scientific concepts, add the actual mathematical formula as comment or to the Doctstrings. This will help maintain a high level of scientific accuracy. E.g. How is are the random walk tracks computed for tropical cyclones?

21.2.6 Pythonic Code

In Python, there are certain structures that are specific to the language, or at least the syntax of how to use them. This is usually referred to as "pythonic" code.

There is an extensive overview on on crucial "pythonic" structures and methods in the Python 101 library.

A few important examples are:

- iterables such as dictionaries, tuples, lists
- iterators and generators (a very useful construct when it comes to code performance, as the implementation of generators avoids reading into memory huge iterables at once, and allows to read them lazily on-the-go; see this blog post for more details)
- f-strings ("formatted string literals," have an f at the beginning and curly braces containing expressions that will be replaced with their values:

```
>>> name = "Eric"
>>> age = 74
>>> f"Hello, {name}. You are {age}."
'Hello, Eric. You are 74.'
```

- decorators (a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure). Something like:

```
@uppercase_decorator
def say_hi():
    return 'hello there'
```

- type checking (Python is a dynamically typed language; also: cf. "Duck typing". Yet, as a best practice, variables should not change type once assigned)
- Do not use mutable default arguments in your functions (e.g. lists). For example, if you define a function as such:

```
def function(x, list=[]):
    default_list.append(x)
```

  Your list will be mutated for future calls of the functions too. The correct implementation would be the following:

```
def func(x, list=None):
 list = [] if list is None
```

- lambda functions (little, anonymous functions, sth like `high_ord_func(2, lambda x: x * x)`)
- list comprehensions (a short and possibly elegant syntax to create a new list in one line, sth like `newlist = [x for x in range(10) if x < 5]` returns `[0, 1, 2, 3, 4]`)

It is recommended to look up the above concepts in case not familiar with them.

21.3 Commenting & Documenting

21.3.1 What is what

Comments are for developers. They describe parts of the code where necessary to facilitate the understanding of programmers. They are marked by putting a `#` in front of every comment line (for multi-liners, wrapping them inside triple double quotes `"""` is basically possible, but discouraged to not mess up with docstrings). A *documentation string (docstring)* is a string that describes a module, function, class, or method definition. The docstring is a special attribute of the object (`object.__doc__`) and, for consistency, is surrounded by triple double quotes (`"""`). This is also where elaboration of the scientific foundation (explanation of used formulae, etc.) should be documented.

A few general rules:

- Have a look at this blog-post on commenting basics
- Comments should be D.R.Y ("Don't Repeat Yourself.")
- Obvious naming conventions can avoid unnecessary comments (cf. `families_by_city[city]` vs. `my_dict[p]`)
- comments should rarely be longer than the code they support
- All public methods need a doc-string. See below for details on the convention used within the climada project.
- Non-public methods that are not immediately obvious to the reader should at least have a short comment after the def line:

```
def complicated_function(s):
    # This function does something complicated
```

21.3.2 Numpy-style docstrings

Full reference can be found here. The standards are such that they use re-structured text (reST) syntax and are rendered using Sphinx.

There are several sections in a docstring, with headings underlined by hyphens (`---`). The sections of a function's docstring are:

1. *Short summary*: A one-line summary that does not use variable names or the function name

```
def add(a, b):
    """The sum of two numbers.

    """
```

2. *Deprecation warning* (use if applicable): to warn users that the object is deprecated, including version the object that was deprecated, and when it will be removed, reason for deprecation, new recommended way of obtaining the same functionality. Use the `deprecated` Sphinx directive:

```
.. deprecated:: 1.6.0
          `ndobj_old` will be removed in NumPy 2.0.0, it is replaced by
          `ndobj_new` because the latter works also with array subclasses.
```

3. *Extended Summary*: A few sentences giving an extended description to clarify functionality, not to discuss implementation detail or background theory (see `Notes` section below!)

4. *Parameters*:
   ```
   Parameters
   ----------
   x : type
       Description of parameter `x`.
   ```
   Description of the function arguments, keywords and their respective types. Enclose variables in single backticks in the description. The colon must be preceded by a space, or omitted if the type is absent. For the parameter types, be as precise as possible. If it is not necessary to specify a keyword argument, use `optional` after the type specification: e.g. `x: int, optional`. Default values of optional parameters can also be detailed in the description. (e.g. `... description of parameter ... (default is -1)`)

5. *Returns*: Explanation of the returned values and their types. Similar to the Parameters section, except the name of each return value is optional, type isn't. If both the name and type are specified, the Returns section takes the same form as the Parameters section.
   ```
   Returns
   -------
   err_code : int
       Non-zero value indicates error code, or zero on success.
   err_msg : str or None
       Human readable error message, or None on success.
   ```

There is a range of other sections that can be included, if sensible and applicable, such as `Yield` (for generator functions only), `Raises` (which errors get raised and under what conditions), `See also` (refer to related code), `Notes` (additional information about the code, possibly including a discussion of the algorithm; may include mathematical equations, written in LaTeX format), `References`, `Examples`(to illustrate usage).

21.4 Importing

General remarks

- Imports should be grouped in the following order:
  - Standard library imports (such as `re`, `math`, `datetime`, cf. here)
  - Related third party imports (such as `numpy`)
  - Local application/library specific imports (such as `climada.hazard.base`)
- You should put a blank line between each group of imports.
- Don't introduce new dependencies (library imports) when the desired functionality is already covered by existing dependencies.

Avoid circular importing!!

Circular imports are a form of circular dependencies that are created with the import statement in Python; e.g. module A loads a method in module B, which in turn requires loading module A. This can generate problems such as tight coupling between modules, reduced code reusability, more difficult maintenance. Circular dependencies can be the source of potential failures, such as infinite recursions, memory leaks, and cascade effects. Generally, they can be resolved with better code design. Have a look here for tips to identify and resolve such imports.

Varia

- there are absolute imports (uses the full path starting from the project's root folder) and relative imports (uses the path starting from the current module to the desired module; usually in the for `from .<module/package> import X`; dots `.` indicate how many directories upwards to traverse. A single dot corresponds to the current directory; two dots indicate one folder up; etc.)
- generally try to avoid star imports (e.g. `from packagename import *`)

Importing utility functions

When importing CLIMADA utility functions (from climada.util), the convention is to import the function as "u_name_of_function", e.g.:

```
from climada.util import coordinates as u_coord
u_coord.make_map()
```

21.5 How to structure a method or function

To clarify ahead: The questions of how to structure an entire module, or even "just" a class, are not treated here. For this, please get in contact with the repository admins to help you go devise a plan.

The following few principles should be adhered to when designing a function or method (which is simply the term for a function inside a class):

- have a look at this blog-post summarizing a few important points to define your function (key-words *abstraction, reusability, modularity*)
- separate algorithmic computations and data curation
- adhere to a maximum method length (rule of thumb: if it doesn't fit your screen, it's probably an indicator that you should refactor into sub-functions)
- divide functions into single purpose pieces (one function, one goal)

21.6 Debugging

When writing code, you will encounter bugs and hence go through (more or less painful) debugging. Depending on the IDE you use, there are different debugging tools that will make your life much easier. They offer functionalities such as stopping the execution of the function just before the bug occurs (via breakpoints), allowing to explore the state of defined variables at this moment of time.

For spyder specifically, have a look at the instructions on how to use ipdb

CHAPTER

TWENTYTWO

EXCEPTION HANDLING AND LOGGING

Exception handling and logging are two important components of programming, in particular for debugging purposes. Detailed technical guides are available online (e.g., Loggin, Error and Exceptions). Here we only repeat a few key points and list a few guidelines for CLIMADA.

22.1 Exception handling

22.1.1 CLIMADA guidelines

1. Catch specific exceptions if possible, i.e, if not needed do not catch all exceptions.
2. Do not catch exception if you do not handle them.
3. Make a clear explanatory message when you raise an error (similarly to when you use the logger to inform the user). Think of future users and how it helps them understanding the error and debugging their code.
4. Catch an exception when it arises.
5. When you catch an exception and raise an error, it is in often (but not always) a good habit to not throw away the first caught exception as it may contain useful information for debugging. (use `raise Error from`)

```
#Bad (1)
x = 1
try:
    l = len(events)
    if l < 1:
        print("l is too short")
except:
    pass
```

```
#Still bad (2)
try:
    l = len(events)
    if l < 1:
        print("l is too short")
except TypeError:
    pass
```

```
#Better, but still unsufficient (3)
try:
    l = len(events)
    if l < 1:
```

(continues on next page)

(continued from previous page)

```
        raise ValueError("To compute an impact there must be at least one event.")
except TypeError:
    raise TypeError("The provided variable events is not a list")
```

```
#Even better (4)
try:
    l = len(events)
except TypeError:
    raise TypeError("The provided variable events is not a list")
if l < 1:
    raise ValueError("To compute an impact there must be at least one event.")
```

```
#Even better (5)
try:
    l = len(events)
except TypeError as tper:
    raise TypeError("The provided variable events is not a list") from tper
if l < 1:
    raise ValueError("To compute an impact there must be at least one event.")
```

22.1.2 Exceptions reminder

Why do we bother to handle exceptions?

- The most essential benefit is to inform the user of the error, while still allowing the program to proceed.

22.2 Logging

22.2.1 CLIMADA guidelines

- In CLIMADA, you cannot use printing. Any output must go into the LOGGER.
- For any logging messages, always think about the audience. What would a user or developer need for information? This also implies to carefully think about the correct LOGGER level. For instance, some information is for debugging, then use the debug level. In this case, make sure that the message actually helps the debugging process! Some message might just inform the user about certain default parameters, then use the inform level. See below for more details about logger levels.
- Do not overuse the LOGGER. Think about which level of logging. Logging errors must be useful for debugging.

You can set the level of the LOGGER using `climada.util.config.LOGGER.setLevel(logging.XXX)`. This way you can for instance 'turn-off' info messages when you are making an application. For example, setting the logger to the "ERROR" level, use:

```
import logging
from climada.util.config import LOGGER
LOGGER.setLevel(logging.ERROR)
```

What levels to use in CLIMADA?

- Debug: what you would print while developing/debugging
- Info: information for example in the check instance

- Warning: whenever CLIMADA fills in values, makes an extrapolation, computes something that might potentially lead to unwanted results (e.g., the 250year damages extrapolated from data over 20 years)

No known use case:

- Error: instead, raise an Error and add the message (raise ValueError("Error message"))
- Critical: …

22.2.2 Reminder about Logging

"Logging is a means of tracking events that happen when some software runs."

When to use logging

"Logging provides a set of convenience functions for simple logging usage. These are debug(), info(), warning(), error() and critical(). To determine when to use logging, see the table below, which states, for each of a set of common tasks, the best tool to use for it."

Task you want to perform	The best tool for the task
Display console output for ordinary usage of a command line script or program	`print()`
Report events that occur during normal operation of a program (e.g. for status monitoring or fault investigation)	`logging.info()` (or `logging.debug()` for very detailed output for diagnostic purposes)
Issue a warning regarding a particular runtime event	`warnings.warn()` in library code if the issue is avoidable and the client application should be modified to eliminate the warning `logging.warning()` if there is nothing the client application can do about the situation, but the event should still be noted
Report an error regarding a particular runtime event	Raise an exception
Report suppression of an error without raising an exception (e.g. error handler in a long-running server process)	`logging.error()`, `logging.exception()` or `logging.critical()` as appropriate for the specific error and application domain

Logger level

"The logging functions are named after the level or severity of the events they are used to track. The standard levels and their applicability are described below (in increasing order of severity):"

Level	When it's used
`DEBUG`	Detailed information, typically of interest only when diagnosing problems.
`INFO`	Confirmation that things are working as expected.
`WARNING`	An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.
`ERROR`	Due to a more serious problem, the software has not been able to perform some function.
`CRITICAL`	A serious error, indicating that the program itself may be unable to continue running.

CHAPTER

TWENTYTHREE

PYTHON PERFORMANCE TIPS AND BEST PRACTICE FOR CLIMADA DEVELOPERS

This guide covers the following recommendations:

□ **Use profiling tools** to find and assess performance bottlenecks.
□ **Replace for-loops** by built-in functions and efficient external implementations.
□ **Consider algorithmic performance**, not only implementation performance.
□ **Get familiar with NumPy:** vectorized functions, slicing, masks and broadcasting.
● **Miscellaneous:** sparse arrays, Numba, parallelization, huge files (xarray), memory.
⚠ **Don't over-optimize** at the expense of readability and usability.

23.1 Profiling

Python comes with powerful packages for the **performance assessment** of your code. Within IPython and notebooks, there are several magic commands for this task:

- `%time`: Time the execution of a single statement
- `%timeit`: Time repeated execution of a single statement for more accuracy
- `%%timeit` Does the same as `%timeit` for a whole cell
- `%prun`: Run code with the profiler
- `%lprun`: Run code with the line-by-line profiler
- `%memit`: Measure the memory use of a single statement
- `%mprun`: Run code with the line-by-line memory profiler

More information on profiling in the Python Data Science Handbook.

Also useful: unofficial Jupyter extension Execute Time.

While it's easy to assess how fast or slow parts of your code are, including finding the bottlenecks, **generating an improved version of it is much harder**. This guide is about **simple best practices** that everyone should know who works with Python, especially when models are performance-critical.

In the following, we will **focus on arithmetic operations** because they play an important role in CLIMADA. Operations on non-numeric objects like strings, graphs, databases, file or network IO might be just as relevant inside and outside of the CLIMADA context. Some of the tips presented here do also apply to other contexts, but **it's always worth looking for context-specific performance guides**.

23.2 General considerations

This section will be concerned with:

 for-loops and built-ins
 external implementations and converting data structures
 algorithmic efficiency
 memory usage

Σ As this section's toy example, let's assume we want to sum up all the numbers in a list:

```
list_of_numbers = list(range(10000))
```

23.2.1 for-loops

A developer with a background in C++ would probably loop over the entries of the list:

```
%%timeit
result = 0
for i in list_of_numbers:
    result += i
```

```
332 µs ± 65.7 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

The built-in function `sum` is much faster:

```
%timeit sum(list_of_numbers)
```

```
54.9 µs ± 5.63 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

The timing improves by a factor of 5-6 and this is not a coincidence: **for-loops generally tend to get prohibitively expensive** when the number of iterations increases.

** When you have a for-loop with many iterations in your code, check for built-in functions or efficient external implementations of your programming task.**

A special case worth noting are `append` operations on lists which can often be replaced by more efficient *list comprehensions*.

23.2.2 Converting data structures

** When you find an external library that solves your task efficiently, always consider that it might be necessary to convert your data structure which takes time.**

For arithmetic operations, NumPy is a great library, but if your data comes as a Python list, NumPy will spend quite some time converting it to a NumPy array:

```
import numpy as np
%timeit np.sum(list_of_numbers)
```

```
572 µs ± 80 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

This operation is even slower than the for-loop!

However, if you can somehow obtain your data in the form of **NumPy arrays from the start**, or if you perform many operations that might compensate for the conversion time, the gain in performance can be considerable:

```
# do the conversion outside of the `%timeit`
ndarray_of_numbers = np.array(list_of_numbers)
%timeit np.sum(ndarray_of_numbers)
```

```
10.6 µs ± 1.56 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Indeed, this is 5-6 times faster than the built-in `sum` and 20-30 times faster than the for-loop.

23.2.3 Always consider several implementations

Even for such a basic task as summing, there exist several implementations whose performance can vary more than you might expect:

```
%timeit ndarray_of_numbers.sum()
%timeit np.einsum("i->", ndarray_of_numbers)
```

```
9.07 µs ± 1.39 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
5.55 µs ± 383 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

This is up to 50 times faster than the for-loop. More information about the `einsum` function will be given in the NumPy section of this guide.

23.2.4 Efficient algorithms

Consider algorithmic performance, not only implementation performance.

All of the examples above do exactly the same thing, algorithmically. However, often the largest performance improvements can be obtained from **algorithmic changes**. This is the case when your model or your data contain symmetries or more complex structure that allows you to skip or boil down arithmetic operations.

In our example, we are summing the numbers from 1 to 10,000 and it's a well known mathematical theorem that this can be done using only two multiplications and an increment:

```
n = max(list_of_numbers)
%timeit 0.5 * n * (n + 1)
```

```
83.1 ns ± 2.5 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
```

Not surprisingly, This is almost 100 times faster than even the fastest implementation of the 10,000 summing operations listed above.

You don't need a degree in maths to find algorithmic improvements. Other algorithmic improvements that are often easy to detect are:

- **Filter your data set as much as possible** to perform operations only on those entries that are really relevant. ***Example:*** When computing a physical hazard (e.g. extreme wind) with CLIMADA, restrict to Centroids on land unless you know that some of your exposure is off shore.
- Make sure to **detect inconsistent or trivial input parameters early on**, before starting any operations. ***Example:*** If your code does some complicated stuff and applies a user-provided normalization factor at the very end, make sure to check that the factor is not 0 before you start applying those complicated operations.

 In general: Before starting to code, take pen and paper and write down what you want to do from an algorithmic perspective.

23.2.5 Memory usage

 Be careful with deep copies of large data sets and only load portions of large files into memory as needed.

Write your code in such a way that you **handle large amounts of data chunk by chunk** so that Python does not need to load everything into memory before performing any operations. When you do, Python's generators might help you with the implementation.

 Allocating unnecessary amounts of memory might slow down your code substantially due to swapping.

23.3 NumPy-related tips and best practice

As mentioned above, arithmetic operations in Python can profit a lot from NumPy's capabilities. In this section, we collect some tips how to make use of NumPy's capabilities when performance is an issue.

23.3.1 Vectorized functions

We mentioned above that Python's **for-loops are really slow**. This is even more important when looping over the entries in a NumPy array. Fortunately, NumPy's masks, slicing notation and vectorization capabilities help to avoid for-loops in almost every possible situation:

```
# TASK: compute the column-sum of a 2-dimensional array
input_arr = np.random.rand(100, 3)
```

```
%%timeit
# SLOW: summing over columns using loops
output = np.zeros(100)
for row_i in range(input_arr.shape[0]):
    for col_i in range(input_arr.shape[1]):
        output[row_i] += input_arr[row_i, col_i]
```

```
145 µs ± 5.47 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

```
# FASTER: using NumPy's vectorized `sum` function with `axis` attribute
%timeit output = input_arr.sum(axis=1)
```

```
4.23 µs ± 216 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In the special case of multiplications and sums (linear operations) over the axes of two multi-dimensional arrays, NumPy's **einsum** is even faster:

```
%timeit output = np.einsum("ij->i", input_arr)
```

```
2.38 µs ± 214 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Another `einsum` example: **Euclidean norms**

```
many_vectors = np.random.rand(1000, 3)
%timeit np.sqrt((many_vectors**2).sum(axis=1))
%timeit np.linalg.norm(many_vectors, axis=1)
%timeit np.sqrt(np.einsum("...j,...j->...", many_vectors, many_vectors))
```

```
24.4 µs ± 2.18 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
26.5 µs ± 2.44 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
9.5 µs ± 91.1 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

For more information about the capabilities of NumPy's `einsum` function, refer to the official NumPy documentation. However, note that future releases of NumPy will eventually improve the performance of core functions, so that `einsum` will become an example of over-optimization (see above) at some point. Whenever you use `einsum`, consider adding a comment that explains what it does for users that are not familiar with `einsum`'s syntax.

Not only `sum`, but many NumPy functions come with similar vectorization capabilities. You can take minima, maxima, means or standard deviations along selected axes. But did you know that the same is true for the `diff` and `argmin` functions?

```
arr = np.random.randint(low=0, high=10, size=(4, 3))
arr
```

```
array([[4, 2, 6],
       [2, 3, 4],
       [3, 3, 3],
       [3, 2, 4]])
```

```
arr.argmin(axis=1)
```

```
array([1, 0, 0, 1])
```

23.3.2 Broadcasting

When operations are performed on several arrays, possibly of differing shapes, be sure to use NumPy's **broadcasting** capabilities. This will save you a lot of memory and time when performing arithmetic operations.

Example: We want to multiply the columns of a two-dimensional array by values stored in a one-dimensional array. There are two naive approaches to this:

```
input_arr = np.random.rand(100, 3)
col_factors = np.random.rand(3)
```

```
# SLOW: stack/tile the one-dimensional array to be two-dimensional
%timeit output = np.tile(col_factors, (input_arr.shape[0], 1)) * input_arr
```

```
5.67 µs ± 718 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
%%timeit
# SLOW: loop over columns and factors
output = input_arr.copy()
for i, factor in enumerate(col_factors):
    output[:, i] *= factor
```

```
9.63 µs ± 95.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

The idea of *broadcasting* is that NumPy **automatically matches axes from right to left and implicitly repeats data along missing axes** if necessary:

```
%timeit output = col_factors * input_arr
```

```
1.41 µs ± 51.7 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

For automatic broadcasting, the *trailing* dimensions of two arrays have to match. NumPy is matching the shapes of the arrays *from right to left*. If you happen to have arrays where other dimensions match, **you have to tell NumPy which dimensions to add by adding an axis of length 1 for each missing dimension**:

```
input_arr = np.random.rand(3, 100)
row_factors = np.random.rand(3)
output = row_factors.reshape(3, 1) * input_arr
```

Because this concept is so important, there is a short-hand notation for adding an axis of length 1. In the slicing notation, **add `None` in those positions where broadcasting should take place**.

```
input_arr = np.random.rand(3, 100)
row_factors = np.random.rand(3)
output = row_factors[:, None] * input_arr
```

```
input_arr = np.random.rand(7, 3, 5, 4, 6)
factors = np.random.rand(7, 3, 4)
output = factors[:, :, None, :, None] * input_arr
```

23.3.3 A note on in-place operations

While **in-place operations** are generally faster than long and explicit expressions, they shouldn't be over-estimated when looking for performance bottlenecks. Often, the loss in code readability is not justified because NumPy's memory management is really fast.

Don't over-optimize!

```
shape = (1200, 1700)
arr_a = np.random.rand(*shape)
arr_b = np.random.rand(*shape)
arr_c = np.random.rand(*shape)
```

```
# long expression in one line
%timeit arr_d = arr_c * (arr_a + arr_b) - arr_a + arr_c
```

```
17.3 ms ± 820 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
%%timeit
# almost same performance: in-place operations
arr_d = arr_a + arr_b
arr_d *= arr_c
arr_d -= arr_a
arr_d += arr_c
```

```
17.4 ms ± 618 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
# You may want to install the module "memory_profiler" first: activate the␣
↪environment climada_env in an Anaconda prompt,
# type "pip install memory_profiler" and execute it
%load_ext memory_profiler
```

```
# long expression in one line
%memit arr_d = arr_c * (arr_a + arr_b) - arr_a + arr_c
```

```
peak memory: 156.68 MiB, increment: 31.20 MiB
```

```
%%memit
# almost same memory usage: in-place operations
arr_d = arr_a + arr_b
arr_d *= arr_c
arr_d -= arr_a
arr_d += arr_c
```

```
peak memory: 157.27 MiB, increment: 0.00 MiB
```

23.4 Miscellaneous

23.4.1 Sparse matrices

In many contexts, we deal with sparse matrices or sparse data structures, i.e. two-dimensional arrays where most of the entries are 0. In CLIMADA, this is especially the case for the `intensity` attributes of `Hazard` objects. This kind of data is usually handled using SciPy's submodule scipy.sparse.

□ When dealing with sparse matrices make sure that you always understand exactly which of your variables are sparse and which are dense and only switch from sparse to dense when absolutely necessary.

□ Multiplications (`multiply`) and matrix multiplications (`dot`) are often faster than operations that involve masks or indexing.

As an example for the last rule, consider the problem of multiplying certain rows of a sparse array by a scalar:

```
import scipy.sparse as sparse

array = np.tile(np.array([0, 0, 0, 2, 0, 0, 0, 1, 0], dtype=np.float64), (100, 80))
row_mask = np.tile(np.array([False, False, True, False, True], dtype=bool), (20,))
```

In the following cells, note that the code in the first line after the `%%timeit` statement is not timed, it's the setup line.

```
%%timeit sparse_array = sparse.csr_matrix(array)
sparse_array[row_mask, :] *= 5
```

```
/home/tovogt/.local/share/miniconda3/envs/tc/lib/python3.7/site-packages/scipy/sparse/
↪data.py:55: RuntimeWarning: overflow encountered in multiply
  self.data *= other
```

```
1.52 ms ± 155 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
%%timeit sparse_array = sparse.csr_matrix(array)
sparse_array.multiply(np.where(row_mask, 5, 1)[:, None]).tocsr()
```

```
340 µs ± 7.32 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
%%timeit sparse_array = sparse.csr_matrix(array)
sparse.diags(np.where(row_mask, 5, 1)).dot(sparse_array)
```

```
400 µs ± 6.43 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

23.4.2 Fast for-loops using Numba

As a last resort, if there's no way to avoid a for-loop even with NumPy's vectorization capabilities, you can use the `@njit` decorator provided by the Numba package:

```
from numba import njit

@njit
def sum_array(arr):
    result = 0.0
    for i in range(arr.shape[0]):
        result += arr[i]
    return result
```

In fact, the Numba function is more than 100 times faster than without the decorator:

```
input_arr = np.float64(np.random.randint(low=0, high=10, size=(10000,)))
```

```
%timeit sum_array(input_arr)
```

```
10.9 µs ± 444 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
# Call the function without the @njit
%timeit sum_array.py_func(input_arr)
```

```
1.84 ms ± 65.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

However, whenever available, **NumPy's own vectorized functions will usually be faster than Numba**.

```
%timeit np.sum(input_arr)
%timeit input_arr.sum()
%timeit np.einsum("i->", input_arr)
```

```
7.6 µs ± 687 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
5.27 µs ± 411 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
7.89 µs ± 499 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

� Make sure you understand the basic idea behind Numba before using it, read the Numba docs.

� Don't use `@jit`, but use `@njit` which is an alias for `@jit(nopython=True)`.

When you know what you are doing, the `fastmath` and `parallel` options can boost performance even further: read more about this in the Numba docs.

23.4.3 Parallelizing tasks

Depending on your hardware setup, parallelizing tasks using pathos and Numba's automatic parallelization feature can improve the performance of your implementation.

□ **Expensive hardware is no excuse for inefficient code.**

Many tasks in CLIMADA could profit from GPU implementations. However, currently there are **no plans to include GPU support in CLIMADA** because of the considerable development and maintenance workload that would come with it. If you want to change this, contact the core team of developers, open an issue or mention it in the bi-weekly meetings.

23.4.4 Read NetCDF datasets with `xarray`

When dealing with NetCDF datasets, memory is often an issue, because even if the file is only a few megabytes in size, the uncompressed raw arrays contained within can be several gigabytes large (especially when data is sparse or similarly structured). One way of dealing with this situation is to open the dataset with `xarray`.

□ **`xarray` allows to read the shape and type of variables contained in the dataset without loading any of the actual data into memory.**

Furthermore, when loading slices and arithmetically aggregating variables, memory is allocated not more than necessary, but values are obtained on-the-fly from the file.

23.5 Take-home messages

We conclude by repeating the gist of this guide:

□ **Use profiling tools** to find and assess performance bottlenecks.
□ **Replace for-loops** by built-in functions and efficient external implementations.
□ **Consider algorithmic performance**, not only implementation performance.
□ **Get familiar with NumPy:** vectorized functions, slicing, masks and broadcasting.
● **Miscellaneous:** sparse arrays, Numba, parallelization, huge files (xarray), memory.
⚠ **Don't over-optimize** at the expense of readability and usability.

CHAPTER

TWENTYFOUR

CLIMADA CODING CONVENTIONS

24.1 Dependencies (python packages)

Python is extremely powerful thanks to the large amount of available libraries, packages and modules. However, maintaining a code with a large number of such packages creates dependencies which is very care intensive. Indeed, each package developer can and does update and develop continuously. This means that certain code can become obsolete over time, stop working altogether, or become incompatible with other packages. Hence, it is crucial to keep the philosophy:

As many packages as needed, as few as possible.

Thus, when you are coding, follow these priorities:

1. Python standard library
2. Functions and methods already implemented in CLIMADA (do NOT introduce circulary imports though)
3. Packages already included in CLIMADA
4. Before adding a new dependency:
   - Contact a repository admin to get permission
   - Open an issue

Hence, first try to solve your problem with the standard library and function/methods already implemented in CLIMADA (see in particular the *utility functions*) then use the packages included in CLIMADA, and if this is not enough, propose the addition of a new package. Do not hesitate to propose new packages if this is needed for your work!

24.2 Class inheritance

In Python, a class can inherit from other classes, which is a very useful mechanism in certain circumstances. However, it is wise to think about inheritance before implementing it. Very important, is that CLIMADA classes do not inherit from external library classes. For example, Exposure directly inherited from Geopandas. This caused problems in CLIMADA when the package Geopandas was updated.

CLIMADA classes shall NOT inherit classes from external modules.

24.3 Paper repository

Applications made with CLIMADA which are published in the form of a paper or a report are very much encouraged to be submitted to the climada/paper repository. You can either:

- Prepare a well-commented jupyter notebook with the code necessary to reproduce your results and upload it to the climada/paper repository. Note however that the repository cannot be used for storing data files.
- Upload the code necessary to reproduce your results to a separate repository of your own. Then, add a link to your repository and to your publication to the readme file on the climada/paper repository.

Notes about DOI

Some journals require you to provide a DOI to the code and data used for your publication. In this case, we encourage you to create a separate repository for your code and create a DOI using Zenodo or any specific service from your institution (e.g. ETH Zürich).

The CLIMADA releases are also identified with a DOI.

24.4 Utility functions

In CLIMADA, there is a set of utility functions defined in `climada.util`. A few examples are:

- convert large monetary numbers into thousands, millions or billions together with the correct unit name
- compute distances
- load hdf5 files
- convert iso country numbers between formats
- …

Whenever you develop a module or make a code review, be attentive to see whether a given functionality has already been implemented as a utility function. In addition, think carefully whether a given function/method does belong in its module or is actually independent of any particular module and should be defined as a utility function.

It is very important to not reinvent the wheel and to avoid unnecessary redundancies in the code. This makes maintenance and debugging very tedious.

24.5 Data dependencies

24.5.1 Web APIs

CLIMADA relies on open data available through web APIs such as those of the World Bank, Natural Earth, NASA and NOAA. You might execute the test `climada_python-x.y.z/test_data_api.py` to check that all the APIs used are active. If any is out of service (temporarily or permanently), the test will indicate which one.

24.5.2 Manual download

As indicated in the software and tutorials, other data might need to be downloaded manually by the user. The following table shows these last data sources, their version used, its current availability and where they are used within CLIMADA:

Name	Version	Link	CLIMADA class	CLIMADA version	CLIMADA tutorial reference
Fire Information for Resource Management System		FIRMS	BushFire	> v1.2.5	climada_hazard_BushFire.ipynb
Gridded Population of the World (GPW)	v4.11	GPW4.1	LitPop	> v1.2.3	climada_entity_LitPop.ipynb

24.6 Side note on parameters

24.6.1 Don't use *args and **kwargs parameters without a very good reason.

There *are* valid use cases for this kind of parameter notation.
In particular `*args` comes in handy when there is an unknown number of equal typed arguments to be passed. E.g., the `pathlib.Path` constructor.
But if the parameters are expected to be structured in any way, it is just a bad idea.

```
def f(x, y, z):
    return x + y + z

# bad in most cases
def g(*args, **kwargs):
    x = args[0]
    y = kwargs['y']
    s = f(*args, **kwargs)
    print(x, y, s)

g(1,y=2,z=3)
```

```
1 2 6
```

```
# usually just fine
def g(x, y, z):
    s = f(x, y, z)
    print(x, y, s)

g(1,y=2,z=3)
```

```
1 2 6
```

24.6.2 Decrease the number of parameters.

Though CLIMADA's pylint configuration .pylintrc allows 7 arguments for any method or function before it complains, it is advisable to aim for less. It is quite likely that a function with so many parameters has an inherent design flaw.

There are very well designed command line tools with inumerable optional arguments, e.g., rsync - but these are command line tools. There are also methods like `pandas.DataFrame.plot()` with countless optional arguments and it makes perfectly sense.

But within the climada package it probably doesn't. divide et impera!

Whenever a method has more than 5 parameters, it is more than likely that it can be refactored pretty easily into two or more methods with less parameters and less complexity:

```python
def f(a, b, c, d, e, f, g, h):
    print(f'f does many things with a lot of arguments: {a, b, c, d, e, f, g, h}')
    return sum([a, b, c, d, e, f, g, h])

f(1, 2, 3, 4, 5, 6, 7, 8)
```

```
f does many things with a lot of arguments: (1, 2, 3, 4, 5, 6, 7, 8)
```

```
36
```

```python
def f1(a, b, c, d):
    print(f'f1 does less things with fewer arguments: {a, b, c, d}')
    return sum([a, b, c, d])

def f2(e, f, g, h):
    print(f'f2 dito: {e, f, g, h}')
    return sum([e, f, g, h])

def f3(x, y):
    print(f'f3 dito, but on a higher level: {x, y}')
    return sum([x, y])

f3(f1(1, 2, 3, 4), f2(5, 6, 7, 8))
```

```
f1 does less things with fewer arguments: (1, 2, 3, 4)
f2 dito: (5, 6, 7, 8)
f3 dito, but on a higher level: (10, 26)
```

```
36
```

This of course pleads the case on a strictly formal level. No real complexities have been reduced during the making of this example.
Nevertheless there is the benefit of reduced test case requirements. And in real life, real complexity *will* be reduced.

CHAPTER

TWENTYFIVE

CLIMADA DOCUMENTATION

The CLIMADA documentation consists of `.rst` files and Jupyter Notebooks. It is built into an HTML webpage by the Sphinx package.

The online documentation is automatically built when the `main` or `develop` branch are updated. Additionally, documentation previews will be built for every pull request on GitHub, and will be displayed under "Checks".

Note that the online documentation allows you to switch versions. By default, you will see the `stable` version, which refers to the latest release. You can switch to `latest`, which refers to the latest version of the `develop` branch.

25.1 Local Build

You can also build and browse the documentation on your machine. This can be useful if you want to access the documentation of a particular feature branch or to check your updates to the documentation.

For building the documentation, you need to follow the advanced installation instructions. Make sure to install the developer requirements as well.

Then, activate the `climada_env` and navigate to the `doc` directory:

```
conda activate climada_env
cd climada_python/doc
```

Next, execute `make` (this might take a while when executed for the first time)

```
make html
```

The documentation will be placed in `doc/_html`. Simply open the page `doc/_html/index.html` with your browser.

25.2 Updating the Documentation Environment for Readthedocs.org

The online documentation is built by `readthedocs.org`. Their servers have a limited capacity, which is typically exceeded by Anaconda when it tries to resolve all dependencies for CLIMADA. We therefore provide a dedicated environment with *fixed* package versions in `requirements/env_docs.yml`.

For re-creating this environment, we provide a Dockerfile. You can use it to build a new environment and extract the exact versions from it. This might be necessary when we upgrade to a new version of Python, or when dependencies are updated. **NOTE:** Your machine must be able to run/virtualize an AMD64 OS.

Follow these instructions:

1. Install Docker on your machine.

2. Enter the top-level directory of the CLIMADA repository with your shell:

```
cd climada_python
```

3. Instruct Docker to build an image from the `doc/create_env_doc.dockerfile`:

```
docker build -f doc/create_env_doc.dockerfile -t climada_env_doc ./
```

4. Run a container from this image:

```
docker run -it climada_env_doc
```

5. You have now entered the container. Activate the conda environment and export its specs:

```
conda activate climada_doc
conda env export
```

   Copy and paste the shell output of the last command into the `requirements/env_docs.yml` file in the CLIMADA repository, overwriting all its contents.

CHAPTER

TWENTYSIX

CLIMADA

CLIMADA stands for **CLIM**ate **ADA**ptation and is a probabilistic natural catastrophe impact model, that also calculates averted damage (benefit) thanks to adaptation measures of any kind (from grey to green infrastructure, behavioural, etc.).

As of today, CLIMADA provides global coverage of major climate-related extreme-weather hazards at high resolution (4x4km) via a data API For select hazards, historic and probabilistic events sets, for past, present and future climate exist at distinct time horizons. You will find a repository containing scientific peer-reviewed articles that explain software components implemented in CLIMADA here.

CLIMADA is divided into two parts (two repositories):

1. the core climada_python contains all the modules necessary for the probabilistic impact, the averted damage, uncertainty and forecast calculations. Data for hazard, exposures and impact functions can be obtained from the data API. Litpop is included as demo Exposures module, and Tropical cyclones is included as a demo Hazard module.
2. the petals climada_petals contains all the modules for generating data (e.g., TC_Surge, WildFire, OpenStreeMap, …). Most development is done here. The petals builds-upon the core and does not work as a stand-alone.

It is recommend for new users to begin with the core (1) and the tutorials therein.

This is the Python (3.9+) version of CLIMADA - please see here for backward compatibility with the MATLAB version.

26.1 Getting started

CLIMADA runs on Windows, macOS and Linux. The released versions of CLIMADA are available from conda-forge. Use the Mamba package manager to install it:

```
mamba install -c conda-forge climada
```

It is **highly recommended** to install CLIMADA into a **separate** Conda environment. See the installation guide for further information.

Follow the tutorials in a Jupyter Notebook to see what can be done with CLIMADA and how.

26.2 Documentation

The online documentation is available on Read the Docs.The documentation of each release version of CLIMADA can be accessed separately through the drop-down menu at the bottom of the left sidebar. Additionally, the version 'stable' refers to the most recent release (installed via `conda`), and 'latest' refers to the latest unstable development version (the `develop` branch).

CLIMADA python:

- online (recommended)
- PDF file
- core Tutorials on GitHub

CLIMADA petals:

- online (recommended)
- PDF file
- petals Tutorials on GitHub

The documentation can also be built locally.

26.3 Citing CLIMADA

See the Citation Guide.

Please use the following logo if you are presenting results obtained with or through CLIMADA:

26.4 Contributing

We welcome any contribution to this repository, be it bugfixes and other code changes and additions, documentation improvements, or tutorial updates.

If you would like to contribute, please refer to our *Contribution Guide*.

26.5 Versioning

We use SemVer for versioning. For the versions available, see the releases on this repository.

26.6 License

CHAPTER

TWENTYSEVEN

CHANGELOG

27.1 4.1.0

Release date: 2024-02-14

27.1.1 Fixed

- Fix `util.coordinates.latlon_bounds` for cases where the specified buffer is very large so that the bounds cover more than the full longitudinal range `[-180, 180]` #839
- Fix `climada.hazard.trop_cyclone` for TC tracks crossing the antimeridian #839

27.2 4.1.0

Release date: 2024-02-14

27.2.1 Dependency Changes

Added:

- `pyproj` >=3.5
- `numexpr` >=2.9

Updated:

- `contextily` >=1.3 → >=1.5
- `dask` >=2023 → >=2024
- `numba` >=0.57 → >=0.59
- `pandas` >=2.1 → >=2.1,<2.2
- `pint` >=0.22 → >=0.23
- `scikit-learn` >=1.3 → >=1.4
- `scipy` >=1.11 → >=1.12
- `sparse` >=0.14 → >=0.15
- `xarray` >=2023.8 → >=2024.1

- overpy =0.6 → =0.7
- peewee =3.16.3 → =3.17.1

Removed:

- proj (in favor of pyproj)

27.2.2 Added

- Convenience method api_client.Client.get_dataset_file, combining get_dataset_info and download_dataset, returning a single file objet. #821
- Read and Write methods to and from csv files for the DiscRates class. *#818*
- Add CalcDeltaClimate to unsequa module to allow uncertainty and sensitivity analysis of impact change calculations #844
- Add function safe_divide in util which handles division by zero and NaN values in the numerator or denominator #844
- Add reset_frequency option for the impact.select() function. #847

27.2.3 Changed

- Update Developer and Installation Guides for easier accessibility by new developers. 808
- Add shapes argument to geo_im_from_array to allow flexible turning on/off of plotting coastline in plot_intensity. #805
- Update CONTRIBUTING.md to better explain types of contributions to this repository #797
- The default tile layer in Exposures maps is not Stamen Terrain anymore, but CartoDB Positron. Affected methods are climada.engine.Impact.plot_basemap_eai_exposure,climada.engine.Impact.plot_basemap_impact_exposure and climada.entity.Exposures.plot_basemap. #798
- Recommend using Mamba instead of Conda for installing CLIMADA #809
- Hazard.from_xarray_raster now allows arbitrary values as 'event' coordinates #837
- climada.test.get_test_file now compares the version of the requested test dataset with the version of climada itself and selects the most appropriate dataset. In this way a test file can be updated without the need of changing the code of the unittest. #822
- Explicitly require pyproj instead of proj (the latter is now implicitly required) #845

27.2.4 Fixed

- Hazard.from_xarray_raster now stores strings as default values for Hazard.event_name #795
- Fix the dist_approx util function when used with method="geosphere" and log=True and points that are very close. #792
- climada.util.yearsets.sample_from_poisson: fix a bug (#819) and inconsistency that occurs when lambda events per year (lam) are set to 1. [#823]
- In the TropCyclone class in the Holland model 2008 and 2010 implementation, a doublecounting of translational velocity is removed #833

- `climada.util.test.test_finance` and `climada.test.test_engine` updated to recent input data from worldbank #841
- Set `nodefaults` in Conda environment specs because `defaults` are not compatible with conda-forge #845
- Avoid redundant calls to `np.unique` in `Impact.impact_at_reg` #848

27.3 4.0.1

Release date: 2023-09-27

27.3.1 Dependency Changes

Added:

- `matplotlib-base` None → >=3.8

Changed:

- `geopandas` >=0.13 → >=0.14
- `pandas` >=1.5,<2.0 → >=2.1

Removed:

- `matplotlib` >=3.7

27.3.2 Changed

- Rearranged file-system structure: `data` directory moved into `climada` package directory. #781

27.3.3 Fixed

- `climada.util.coordinates.get_country_code` bug, occurring with non-standard longitudinal coordinates around the anti-meridian. #770

27.4 4.0.0

Release date: 2023-09-01

27.4.1 Dependency Updates

Added:

- `pytest` #726
- `pytest-cov` #726
- `pytest-subtests` #726
- `unittest-xml-reporting`

Changed:

- `cartopy` >=0.20.0,<0.20.3 → >=0.21
- `cfgrib` >=0.9.7,<0.9.10 → =0.9.9
- `contextily` >=1.0 → >=1.3
- `dask` >=2.25 → >=2023
- `eccodes` [auto] → =2.27
- `gdal` !=3.4.1 → >=3.6
- `geopandas` >=0.8 → >=0.13
- `h5py` >=2.10 → >=3.8
- `haversine` >=2.3 → >=2.8
- `matplotlib` >=3.2,< 3.6 → >=3.7
- `netcdf4` >=1.5 → >=1.6
- `numba` >=0.51,!=0.55.0 → >=0.57
- `openpyxl` >=3.0 → >=3.1
- `pandas-datareader` >=0.9 → >=0.10
- `pathos` >=0.2 → >=0.3
- `pint` >=0.15 → >=0.22
- `proj` !=9.0.0 → >=9.1
- `pycountry` >=20.7 → >=22.3
- `pytables` >=3.6 → >=3.7
- `rasterio` >=1.2.7,<1.3 → >=1.3
- `requests` >=2.24 → >=2.31
- `salib` >=1.3.0 → >=1.4
- `scikit-learn` >=1.0 → >=1.2
- `scipy` >=1.6 → >=1.10
- `sparse` >=0.13 → >=0.14
- `statsmodels` >=0.11 → >=0.14
- `tabulate` >=0.8 → >=0.9
- `tqdm` >=4.48 → >=4.65
- `xarray` >=0.13 → >=2023.5
- `xlrd` >=1.2 → >=2.0
- `xlsxwriter` >=1.3 → >=3.1

Removed:

- `nbsphinx` #712
- `pandoc` #712
- `xmlrunner`

27.4.2 Added

- `Impact.impact_at_reg` method for aggregating impacts per country or custom region #642
- `Impact.match_centroids` convenience method for matching (hazard) centroids to impact objects #602
- `climada.util.coordinates.match_centroids` method for matching (hazard) centroids to GeoDataFrames #602
- 'Extra' requirements `doc`, `test`, and `dev` for Python package #712
- Added method `Exposures.centroids_total_value` to replace the functionality of `Exposures.affected_total_value`. This method is temporary and deprecated. #702
- New method `climada.util.api_client.Client.purge_cache`: utility function to remove outdated files from the local file system to free disk space. (#737)
- New attribute `climada.hazard.Hazard.haz_type`: used for assigning impacts to hazards. In previous versions this information was stored in the now removed `climada.hazard.tag.Tag` class. #736
- New attribute `climada.entity.exposures.Exposures.description`: used for setting the default title in plots from plotting mathods `plot_hexbin` and `plot_scatter`. In previous versions this information was stored in the deprecated `climada.entity.tag.Tag` class. #756
- Added advanced examples in unsequa tutorial for coupled input variables and for handling efficiently the loading of multiple large files #766

27.4.3 Changed

- Improved error messages from `climada.CONFIG` in case of missing configuration values #670
- Refactored `Exposure.assign_centroids` using a new util function `u_coord.match_centroids` #602
- Renamed `climada.util.coordinate.assign_grid_points` to `match_grid_points` and `climada.util.coordinates.assign_coordinates` to `match_coordinates` #602
- Modified the method to disaggregate lines in the `lines_polys_handler` utility module in order to better conserve the total length of all lines on average #679.
- Added test for non-default impact function id in the `lines_polys_handler` #676
- The sigmoid and step impact functions now require the user to define the hazard type. #675
- Improved error messages produced by `ImpactCalc.impact()` in case hazard type is not found in exposures/impf_set #691
- Tests with long runtime were moved to integration tests in `climada/test` #709
- Use `myst-nb` for parsing Jupyter Notebooks for the documentation instead of `nbsphinx` #712
- Installation guide now recommends installing CLIMADA directly via `conda install` #714
- `Exposures.affected_total_value` now takes a hazard intensity threshold as argument. Affected values are only those for which at least one event exceeds the threshold. (previously, all exposures points with an assigned centroid were considered affected). By default the centroids are reassigned. #702 #730
- Add option to pass region ID to `LitPop.from_shape` #720
- Slightly improved performance on `LitPop`-internal computations #720
- Use `pytest` for executing tests #726

- Users can opt-out of the climada specific logging definitions and freely configure logging to their will, by setting the config value `logging.managed` to `false`. #724
- Add option to read additional variables from IBTrACS when using `TCTracks.from_ibtracs_netcdf` #728
- New file format for `TCTracks` I/O with better performance. This change is not backwards compatible: If you stored `TCTracks` objects with `TCTracks.write_hdf5`, reload the original data and store them again. #735
- Add option to load only a subset when reading TC tracks using `TCTracks.from_simulations_emanuel`. #741
- Set `save_mat` to `False` in the `unsequa` module #746
- `list_dataset_infos` from `climada.util.api_client.Client`: the `properties` argument, a `dict`, can now have `None` as values. Before, only strings and lists of strings were allowed. Setting a particular property to `None` triggers a search for datasets where this property is not assigned. #752
- Reduce memory requirements of `TropCyclone.from_tracks` #749
- Support for different wind speed and pressure units in `TCTracks` when running `TropCyclone.from_tracks` #749
- The title of plots created by the `Exposures` methods `plot_hexbin` and `plot_scatter` can be set as a method argument. #756
- Changed the parallel package from Pathos to Multiproess in the unsequa module #763
- Updated installation instructions to use conda for core and petals #776

27.4.4 Fixed

- `util.lines_polys_handler` solve polygon disaggregation issue in metre-based projection #666
- Problem with `pyproj.CRS` as `Impact` attribute, #706. Now CRS is always stored as `str` in WKT format.
- Correctly handle assertion errors in `Centroids.values_from_vector_files` and fix the associated test #768
- Text in `Forecast` class plots can now be adjusted #769
- `Impact.impact_at_reg` now supports impact matrices where all entries are zero #773
- upgrade pathos 0.3.0 -> 0.3.1 issue #761 (for unsequa module #763)
- Fix bugs with pandas 2.0 (iteritems -> items, append -> concat) (fix issue #700 for unsequa module) #763)
- Remove matplotlib styles in unsequa module (fixes issue #758) #763

27.4.5 Deprecated

- `Centroids.from_geodataframe` and `Centroids.from_pix_bounds` #721
- `Impact.tot_value`: Use `Exposures.affected_total_value` to compute the total value affected by a hazard intensity above a custom threshold #702
- `climada.entity.tag.Tag`. #779. The class is not used anymore but had to be kept for reading Exposures HDF5 files that were created with previous versions of CLIMADA.

27.4.6 Removed

- `Centroids.set_raster_from_pix_bounds` #721
- `requirements/env_developer.yml` environment specs. Use 'extra' requirements when installing the Python package instead #712
- The `climada.entitity.tag.Tag` class, together with `Impact.tag`, `Exposures.tag`, `ImpactFuncSet.tag`, `MeasuresSet.tag`, `Hazard.tag` attributes. This may break backwards-compatibility with respect to the files written and read by the `Impact` class. #736, #743, #753, #754, #756, #767, #779
- `impact.tot_value` attribute removed from unsequa module #763

27.5 v3.3.2

Release date: 2023-03-02

27.5.1 Dependency Updates

Removed:

- `pybufrkit` #662

27.6 v3.3.1

Release date: 2023-02-27

27.6.1 Description

Patch-relaese with altered base config file so that the basic installation test passes.

27.6.2 Changed

- The base config file `climada/conf/climada.conf` has an entry for `CONFIG.hazard.test_data`.

27.7 v3.3.0

Release date: 2023-02-17

27.7.1 Dependency Changes

new:

- sparse (>=0.13) for #578

updated:

- **python 3.9** - python 3.8 will still work, but python 3.9 is now the default version for installing climada (#614)
- contextily >=1.0 (no longer restricted to <1.2 as `contextily.sources` has been replaced in #517)
- cartopy >=0.20.0,<0.20.3 (>=0.20.3 has an issue with geographic crs in plots)
- matplotlib >=3.2,<3.6 (3.6 depends on cartopy 0.21)

27.7.2 Added

- `climada.hazard.Hazard.from_xarray_raster(_file)` class methods for reading `Hazard` objects from an `xarray.Dataset`, or from a file that can be read by `xarray`. #507, #589, #652.
- `climada.engine.impact.Impact` objects have new methods `from_hdf5` and `write_hdf5` for reading their data from, and writing it to, H5 files #606
- `climada.engine.impact.Impact` objects has a new class method `concat` for concatenation of impacts based on the same exposures #529.
- `climada.engine.impact_calc`: this module was separated from `climada.engine.impact` and contains the code that dealing with impact *calculation* while the latter focuses on impact *data* #560.
- The classes `Hazard`, `Impact` and `ImpactFreqCurve` have a novel attribute `frequency_unit`. Before it was implicitly set to annual, now it can be specified and accordingly displayed in plots. #532.
- CONTRIBUTING.md #518.
- Changelog based on the CLIMADA release overview and https://keepachangelog.com template #626.

27.7.3 Changed

- The `Impact` calculation underwent a major refactoring. Now the suggested way to run an impact calculation is by `climada.engine.impact_calc.ImpactCalc.impact()`. #436, #527.
- Addition of uncertainty helper methods variables: list of hazard, list of impact function sets, and hazard fraction. This allows to pre-compute hazards or impact function sets from different sources from which one can then sample uniformly. #513
- Full initialization of most Climada objects is now possible (and suggested!) in one step, by simply calling the constructor with all arguments required for coherently filling the object with data: #560, #553, #550, #564, #563, #565, #573, #569, #570, #574, #559, #571, #549, #567, #568, #562.
- It is possible now to set the `fraction` of a `Hazard` object to `None` which will have the same effect as if it were `1` everywhere. This saves a lot of memory and calculation time, #541.
- The online documentation has been completely overhauled: #597, #600, #609, #620, #615, #617, #622, #656.
- Updated installation instructions #644

27.7.4 Fixed

- several antimeridian issues: #524, #551, #613.
- bug in `climada.hazard.Centroids.set_on_land()` when coordinates go around the globe: #542, #543.
- bug in `climada.util.coordinates.get_country_code()` when all coordinates are on sea.
- suppress pointless warnings in plotting functions, #520.
- test coverage improved: #583, #594, #608, #616, #637.
- deprecated features removoed: #517, #535, #566,

27.7.5 Deprecated

- `climada.enginge.impact.Impact.calc()` and `climada.enginge.impact.Impact.calc_impact_yearset()` #436.

CHAPTER TWENTYEIGHT

CLIMADA LIST OF AUTHORS

- Gabriela Aznar-Siguan
- David N. Bresch
- Samuel Eberenz
- Jan Hartman
- Marine Perus
- Thomas Röösli
- Dario Stocker
- Veronica Bozzini
- Tobias Geiger
- Carmen B. Steinmann
- Evelyn Mühlhofer
- Rachel Bungerer
- Inga Sauer
- Samuel Lüthi
- Pui Man Kam
- Simona Meiler
- Alessio Ciullo
- Thomas Vogt
- Benoit P. Guillod
- Chahan Kropf
- Emanuel Schmid
- Chris Fairless
- Jan Wüthrich
- Zélie Standhanske
- Yue Yu
- Lukas Riedel
- Raphael Portmann

- Nicolas Colombi
- Leonie Villiger
- Kam Lam Yeung
- Sarah Hülsen
- Timo Schmid

CHAPTER

TWENTYNINE

CLIMADA CONTRIBUTION GUIDE

We welcome any contribution to CLIMADA and want to express our thanks to everybody who contributes.

29.1 What Warrants a Contribution?

Anything! For orientation, these are some categories of possible contributions we can think of:

- **Technical problems and bugs:** Did you encounter a problem when using CLIMADA? Raise an issue in our repository, providing a description or ideally a code replicating the error. Did you already find a solution to the problem? Please raise a pull request to help us resolve the issue!
- **Documentation and Tutorial Updates:** Found a typo in the documentation? Is a tutorial lacking some information you find important? Simply fix a line, or add a paragraph. We are happy to incorporate you additions! Please raise a pull request!
- **New Modules and Utility Functions:** Did you create a function or an entire module you find useful for your work? Maybe you are not the only one! Feel free to simply raise a pull request for functions that improve, e.g., plotting or data handling. As an entire module has to be carefully integrated into the framework, it might help if you talk to us first so we can design the module and plan the next steps. You can do that by raising an issue or starting a discussion on GitHub.

A good place to start a personal discussion is our monthly CLIMADA developers call. Please contact the lead developers if you want to join.

29.2 Why Should You Contribute?

- You will be listed as author of the CLIMADA repository in the *AUTHORS* file.
- You will improve the quality of the CLIMADA software for you and for everybody else using it.
- You will gain insights into scientific software development.

29.3 Minimal Steps to Contribute

Before you start, please have a look at our Developer Guide.

To contribute follow these steps:

1. Install CLIMADA following the installation instructions for developers.
2. In the CLIMADA repository, create a new feature branch from the latest `develop` branch:

```
git checkout develop && git pull
git checkout -b feature/my-fancy-branch
```

3. Implement your changes and commit them with meaningful and well formatted commit messages.
4. Add unit and integration tests to your code, if applicable.
5. Use Pylint for a static code analysis of your code with CLIMADA's configuration `.pylintrc`:

```
pylint
```

6. Add your name to the *AUTHORS* file.
7. Push your updates to the remote repository:

```
git push --set-upstream origin feature/my-fancy-branch
```

NOTE: Only team members are allowed to push to the original repository. Most contributors are/will be team members. To be added to the team list and get permissions please contact one of the owners. Alternatively, you can fork the CLIMADA repository and add this fork as a new remote to your local repository. You can then push to the fork remote:

```
git remote add fork <your-fork-url>
git push --set-upstream fork feature/my-fancy-branch
```

8. On the CLIMADA-project/climada_python GitHub repository, create a new pull request with target branch `develop`. This also works if you pushed to a fork instead of the main repository. Add a description and explanation of your changes and work through the pull request author checklist provided. Feel free to request reviews from specific team members.
9. After approval of the pull request, the branch is merged into `develop` and your changes will become part of the next CLIMADA release.

29.4 Resources

The CLIMADA documentation provides a Developer Guide. Here's a selection of the commonly required information:

- How to use Git and GitHub for CLIMADA development: Development and Git and CLIMADA
- Coding instructions for CLIMADA: Python Dos and Don'ts, Performance Tips, CLIMADA Conventions
- How to execute tests in CLIMADA: Testing and Continuous Integration

29.5 Pull Requests

After developing a new feature, fixing a bug, or updating the tutorials, you can create a pull request to have your changes reviewed and then merged into the CLIMADA code base. To ensure that your pull request can be reviewed quickly and easily, please have a look at the *Resources* above before opening a pull request. In particular, please check out the Pull Request instructions.

We provide a description template for pull requests that helps you provide the essential information for reviewers. It also contains a checklist for both pull request authors and reviewers to guide the review process.

CHAPTER

THIRTY

CITATION GUIDE

If you use CLIMADA for your work, please cite the appropriate publications. A list of all CLIMADA code related articles is available on Zotero and can be downloaded as single Bibtex file: `climada_publications.bib`

30.1 Publications by Module

If you use specific tools and modules of CLIMADA, please cite the appropriate publications presenting these modules according to the following table:

Module or tool used	Publication to cite
Any	The Zenodo archive of the CLIMADA version you are using
Impact calculations	Aznar-Siguan, G. and Bresch, D. N. (2019): CLIMADA v1: A global weather and climate risk assessment platform, Geosci. Model Dev., 12, 3085–3097, https://doi.org/10.5194/gmd-14-351-2021
Cost-benefit analysis	Bresch, D. N. and Aznar-Siguan, G. (2021): CLIMADA v1.4.1: Towards a globally consistent adaptation options appraisal tool, Geosci. Model Dev., 14, 351–363, https://doi.org/10.5194/gmd-14-351-2021
Uncertainty and sensitivity analysis	Kropf, C. M. et al. (2022): Uncertainty and sensitivity analysis for probabilistic weather and climate-risk modelling: an implementation in CLIMADA v.3.1.0. Geosci. Model Dev. 15, 7177–7201, https://doi.org/10.5194/gmd-15-7177-2022
Lines and polygons exposures or Open Street Map exposures	Mühlhofer, E., et al. (2023): OpenStreetMap for Multi-Faceted Climate Risk Assessments https://eartharxiv.org/repository/view/5615/
LitPop exposures	Eberenz, S., et al. (2020): Asset exposure data for global physical risk assessment. Earth System Science Data 12, 817–833, https://doi.org/10.3929/ethz-b-000409595

Please find the code to reprocduce selected CLIMADA-related scientific publications in our repository of scientific publications.

30.2 Links and Logo

In presentations or other graphical material, as well as in reports etc., where applicable, please add the following logo: `climada_logo_QR.png`:

As key link, please use https://wcr.ethz.ch/research/climada.html, as it provides a brief introduction especially for those not familiar with GitHub.

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y